Brit. J. Phil. Sci. **60** (2009), 765–792

SAD Computers and Two Versions of the Church–Turing Thesis

Tim Button

ABSTRACT

Recent work on hypercomputation has raised new objections against the Church–Turing Thesis. In this paper, I focus on the challenge posed by a particular kind of hypercomputer, namely, SAD computers. I first consider deterministic and probabilistic barriers to the physical possibility of SAD computation. These suggest several ways to defend a Physical version of the Church–Turing Thesis. I then argue against Hogarth's analogy between non-Turing computability and non-Euclidean geometry, showing that it is a non-sequitur. I conclude that the Effective version of the Church–Turing Thesis is unaffected by SAD computation.

Across a series of papers, Mark Hogarth has defined and investigated SAD computers. These are hypercomputers, in the sense that they can carry out calculations that would otherwise require supertasks. The aim of this paper

doi:10.1093/bjps/axp038
Advance Access published on September 11, 2009

is to investigate the significance of SAD computers for two versions of the Church–Turing Thesis.

In his early work, Hogarth ([1994]) suggested that SAD computers might offer counterexamples to the Physical Church–Turing Thesis.[1] Likewise, Davies ([2001]) has suggested that unrestricted physical Turing computation is as fraught with difficulty as physical SAD computation. I contest this, by raising deterministic and probabilistic problems for SAD computation. These problems suggest several ways to defend the Physical Church–Turing Thesis against the challenge that SAD computers pose (Section 2).

In more recent work, Hogarth ([1996], [2004], [2009]) argues that SAD computers show that there is no 'fundamentally distinguished' computer. This leads him to reject the very idea of an Effective Church–Turing Thesis. I argue that Hogarth's argument is a non-sequitur, and that we have good reasons for remaining interested in effective finitary computability. I conclude that the Effective Church–Turing Thesis is untouched by SAD computers (Section 3).

Before proceeding, I should offer a word of caution. In this paper, I discuss several different notions of computability. They divide into two categories: *finitary* computations, which must deliver any output in finitely many steps, and *infinitary* SAD computations, which may take infinitely many steps. Both categories subdivide further into three kinds: purely *formal* computability (e.g., Turing computability), *physical* computability, and *effective* computability. So we have six notions of computability in all. To avoid ambiguity, I shall have to say things like 'effectively finitarily computable' when most other authors would simply say 'recursive'. I trust the reader will not find this too burdensome.

1 SAD Computability

In this section, I shall present the basic idea behind SAD computability. I will start by informally characterizing it, and discussing some of its motivations. I will then present two physical models for SAD computers, due to Davies and Hogarth, and end by summarizing Welch's technical results concerning the extent of SAD computation.

In what follows, I shall generally consider computers that have been programed to test whether a natural number n is a member of some set $S \subseteq \mathbb{N}$. This is for simplicity; to generalize things, we need only suppose that n (finitarily) encodes an m-tuple of natural numbers, and that S is a set of m-tuples of natural numbers.

[1] Hogarth ([1994], p. 129) coined the acronym 'SAD' to abbreviate 'arithmetical sentence deciding'. As Hogarth's computers have been investigated, this acronym has proved misleading: Welch ([2008]) has shown that certain SAD computers can decide *hyper*arithmetical sets (see Section 1.5). Nevertheless, I shall retain the name 'SAD' to describe the kind of infinitary computer that Hogarth has in mind.

1.1 The basic idea of SAD computation

Suppose that we have some two-place finitarily computable predicate ϕ. We can use this to define an ω-sequence of sets:

$$A_0 := \{n : \phi(n, 0)\}$$
$$A_1 := \{n : \phi(n, 1)\}$$
$$A_2 := \{n : \phi(n, 2)\}$$
$$\ldots$$

For any given natural number n, a finitary computer can determine whether n is in A_i; it just follows some algorithm to check whether $\phi(n, i)$ holds or not. But now we define the Σ_1 set:[2]

$$B := \bigcup_i A_i = \{n : (\exists x)\phi(n, x)\}.$$

In general, a finitary computer cannot be relied upon to test whether an arbitrary natural number is in B. We might make our finitary computer run the following program:

```
i = 0
loop until (n is in A_i):
    i = i+1
print "n is in B"
```

If n is in B, this program will eventually output '`n is in B`', and terminate gracefully. Unfortunately, if n is not in B, we will never reach the last line of the code; the computer will loop through an ω-sequence of calculations without terminating.

But suppose that we had some way to stand 'after' the (potential) ω-sequence of calculations. Then we *could* use the preceding code to test whether n is in B. If n is in B, the program will tell us so. If n is not in B, the program will not have outputted anything after the ω-sequence of calculations, and we will be able to infer, from the silence, that n is not in B.

Since B is an arbitrary Σ_1 set and n is an arbitrary natural number, we could use this method to determine whether any natural number is in any Σ_1 set. And what goes for B goes equally well for its complement, $\overline{B}$, which is an arbitrary Π_1 set: we can test whether n is in $\overline{B}$ using the procedure that I have described, adding just a little extra inference to cover the fact that $n \in B$ iff $n \notin \overline{B}$.

[2] A set is Σ_k iff it can be defined using a Σ_k formula. A Σ_k formula starts with an existential quantifier, only contains quantifiers at the beginning of the formula, has k blocks of quantifiers of the same type, and then has a finitarily computable predicate. For example, where F is finitarily computable, this is a Σ_4 formula:

$$(\exists x_1)(\exists x_2)(\forall y_1)(\forall y_2)(\exists z_1)(\forall w_1)(\forall w_2)(\forall w_3)F(x_1, x_2, y_1, y_2, z_1, w_1, w_2, w_3).$$

A Π_k set is defined similarly, but a Π_k formula starts with '$\forall$' rather than '$\exists$'.

Call this infinitary setup a 'SAD_1 computer', since it enables us to decide membership for some of the arithmetical sets, namely, the Π_1 and Σ_1 sets (Hogarth [1994], p. 127; see footnote 1, above).

1.2 Avoiding supertasks

In the procedure just described, I did not need to review all the workings of the infinitely looping computer. Instead, the computer communicates to me by printing (at most) a single finite string, and I only need to check whether it has printed anything once. As such, I do not need to perform any supertask.

This is a good thing, since there are excellent reasons to be sceptical of the use of supertasks in hypercomputation. Suppose that the print command displays a line of text on a computer monitor. Now let us modify the preceding program slightly, so that whenever the computer tests whether n is in A_i, it also prints '`1`' if `i` is odd, and '`0`' if `i` is even, i.e.,

```
i = 0
loop until (n is in $A_i$):
    if (i is odd) then (print "1")
    else (print "0")
    i = i+1
print "n is in B"
```

If n is not in B, then the monitor will be in no determinate state 'after' the ω-sequence of calculations (since the sequence of numerals that appear on the monitor is divergent). This is a computational version of Thomson's Lamp (Thomson [1954], pp. 5–6; this version appears in Earman and Norton [1993], p. 28).

We could block this specific worry by stipulating that the looping computer can only ever print once. But to make certain that we have avoided *all* paradoxical implications, we need to find some way to make the infinitely looping computer seem like an ordinary, non-paradoxical, supertask-free device. If we can do this, then SAD_1 computers, although infinitary, will not require supertasks.

The basic challenge is this. From the perspective of the user, we need to squeeze the computer's ω-sequence of calculations into a finite period of time. Consequently, from the perspective of the user, the frequency of the computer's calculations must increase exponentially. *Prima facie*, this suggests that the computer must perform a supertask.

In the next two subsections, I will outline two physical models of SAD computation that might claim to avoid appeal to supertasks. I will leave discussion of *physical* problems associated with these models to Section 2, and I return to the need to avoid supertasks in Sections 3.4 and 4.

1.3 Davies's model of SAD computation

The first model is Davies's ([2001]). The strategy is to replace the single infinitely looping computer with infinitely many finitary machines. Davies's model can be presented in any dense spacetime, but his model requires the controversial assumption that matter is infinitely divisible.

We first build an ordinary, finitary machine. This machine is programed to build an exponentially smaller finitary machine, which it programs to build an exponentially smaller finitary machine, and so on, so that every machine builds an exponentially smaller successor. This gives us an ω-sequence of finitary machines, packed into a finite region of space. We also construct our machines so that each machine has an exponentially better clockspeed, and exponentially more memory, than its predecessor. Consequently, an ω-sequence of calculations can be squeezed into a finite duration of time.

It is easy to turn this physical configuration into a SAD$_1$ setup. Suppose we want to check whether n is in a Σ_1 set $B = \bigcup_i A_i$. The 0th machine starts the process, by checking whether n is in A_0. If (and only if) the ith machine finds that n is not in A_i, the $(i + 1)$th machine checks whether n is in A_{i+1}.[3] But if the ith machine finds that n is in A_i, the machine passes a signal up the chain, from machine to machine, until it reaches the 0th machine in the chain, which then prints '`n is in B`'. After a fixed finite time period, all the calculations will have been completed, at which point we just check whether '`n is in B`' has been printed or not.

Each machine in the array only does three things at most: test a problem, build the next machine, and send a signal. A SAD computation is therefore performed without any individual component performing a supertask, and so Thomson's Lamp, and its kin, are avoided (Davies [2001], pp. 676–7).

1.4 Hogarth's model of SAD computation

The second model of SAD computation is Hogarth's ([1992], [1994], p. 126, [1996], pp. 91–4, [2004], pp. 681–2, [2009], pp. 281–3). The strategy is to find a relativistic spacetime structure that allows a human user to survey the entire, infinite worldline of some finitary computer. This requires a Malament–Hogarth spacetime, which is a time-oriented differentiable manifold with a Lorentz metric and three essential components (Hogarth [1992], p. 176):

[3] Actually, we need to be slightly subtler; see (Davies [2001], p. 675) for details.

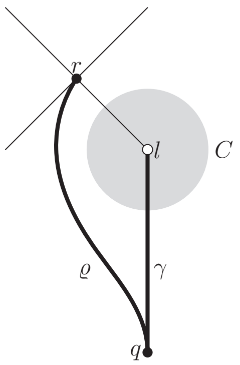

Figure 1. A toy Malament-Hogarth spacetime (see Hogarth [1994], p. 127).

γ: a time-like path with a start-point but no end-point, such that $\int_\gamma d\tau$ is infinite.

r: a point that contains γ in its past.

ϱ: a time-like path, from some point q to r, such that $\int_{\varrho(q,r)} d\tau$ is finite.

By way of illustration, Earman and Norton ([1993], p. 28) construct a toy Malament–Hogarth spacetime from a Minkowski spacetime $\langle \mathbb{R}^4, \eta \rangle$. Choose a compact set of points $C \subset \mathbb{R}^4$ and a point $l \in C$, and define a scalar field Ω that is 1 at every point outside C, and that inside C approaches $+\infty$ as we approach l. Now $\langle \mathbb{R}^4 \setminus \{l\}, \Omega^2\eta \rangle$ is a Malament–Hogarth spacetime, as depicted in Figure 1.

To perform a SAD$_1$ computation in this spacetime, I simply send a computer on the worldline γ, whilst I sit on the worldline ϱ. Suppose that the computer is able to obtain enough energy to continue functioning indefinitely on its worldline, that its memory may grow without limit as it ages, and that its components never wear out. Since $\int_{\varrho(q,r)} d\tau$ is finite, and the computer's eternal history is in the past of point r, the computer can have completed an ω-sequence of calculations after only a finite amount of my proper time has elapsed. So to test whether n is in some Σ_1 set $B = \bigcup_i A_i$, the computer is programed to loop through each A_i, checking whether n is in A_i. If the test is successful, the loop terminates and the computer sends a signal (a beam of light) to me on my worldline. All I need to do is wait until I am at r, and then draw an inference from whether I have received a signal or not.

Again, no component in this setup performs a supertask. In particular, the looping computer does nothing more than plod through routine calculations on its worldline, sending at most one signal to me. 'One is tempted to say the beauty of the setup is that there is no supertask!' (Hogarth [2009], p. 283; see also Hogarth [1996], pp. 78–81, [2004], p. 682; Earman and Norton

[1993], p. 30, [1996], p. 255; Németi and Dávid [2006], Section 5.1; Welch [2008], p. 670).

1.5 Generalizing SAD computers

SAD$_1$ computers are powerful, but they have their limitations: they cannot determine membership of arbitrary Π_2 or Σ_2 sets.[4] Informally, the reason for this limitation is as follows (for formal proofs, see Earman and Norton [1996], pp. 251–4; Hogarth [1994], p. 128). Suppose that we want to use a SAD$_1$ setup to determine whether n is in a Σ_2 set:

$$C := \{n : (\exists x)(\forall y)\psi(n, x, y)\}.$$

The procedure we have used so far would be to ask our machine to loop through the natural numbers, looking for a witness, x, that shows that n is in C. But, for any particular x, a machine cannot 'know' that every y is such that $\psi(n, x, y)$, until it has inspected every y. There is no stage in the loop at which it has done this, so there is no stage at which it can be sure that it possesses a witness for the claim that n is in C.

We overcame the limitations of finitary computation and reached the dizzy heights of SAD$_1$ computation by allowing someone to stand 'after' an ω-sequence of ordinary computations. We can pull the same trick here: we overcome the limitations of a SAD$_1$ computation by allowing someone to stand 'after' an ω-sequence of SAD$_1$ computations. Consider the ω-sequence of Σ_1 sets given by

$$B_i := \{n : (\exists y)\neg\psi(n, i, y)\}$$

and note that $\bigcup_i \overline{B_i} = C$. For each B_i, there is an ω-sequence of finitarily computable sets given by

$$A_{i,j} := \{n : \neg\psi(n, i, j)\}$$

such that $\bigcup_j A_{i,j} = B_i$. So, for each i, we use a SAD$_1$ computer to test whether n is in each B_i, in the normal way. We then chain these setups together to determine whether n is in C. Call this a SAD$_2$ computer.

It is instructive to think of this SAD$_2$ computer as given by the tree of Figure 2. Each node of the tree is associated with a computer performing a

[4] I assumed (Section 1.2) that the looping finitary computer is only allowed to send a single communication to the user. Welch ([2008], pp. 670–1) shows that if the looping computer can send any arbitrary finite number of communications to its user, then a SAD$_1$ computer can determine membership of all and only the $\Pi_2 \cap \Sigma_2$ sets.

Davies ([2001], p. 676) states that his setup, as described in Section 1.3, can test Π_2 sentences, such as the Twin Primes conjecture. This is mistaken: the same restrictions that apply to Hogarth's model apply to Davies's. If Davies's machines can only pass a single signal up the chain, then his computers are restricted to testing arbitrary Σ_1 and Π_1 sets; if his machines can pass arbitrary finite numbers of signals up the chain, then his computers are restricted to testing arbitrary $\Pi_2 \cap \Sigma_2$ sets.

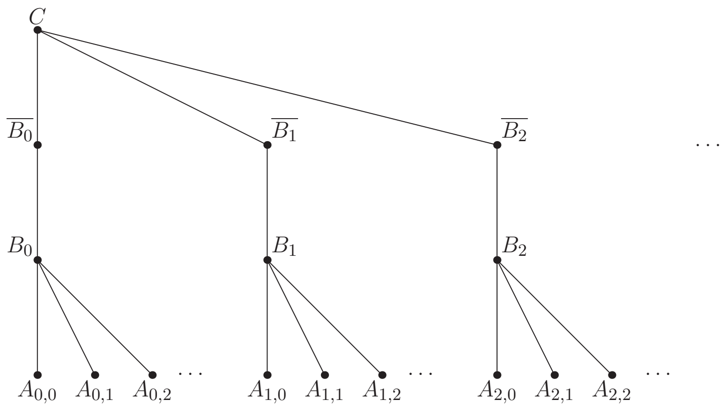
C
$\overline{B_0}$
$\overline{B_1}$
$\overline{B_2}$
B_0
B_1
B_2
$A_{0,0}$ $A_{0,1}$ $A_{0,2}$
$A_{1,0}$ $A_{1,1}$ $A_{1,2}$
$A_{2,0}$ $A_{2,1}$ $A_{2,2}$

Figure 2. A SAD$_2$ computational tree.

test for set membership, and the lines of the tree are thought of as possible signalling paths. In more detail:

$A_{i,j}$: Each node labelled '$A_{i,j}$' is associated with a computer testing whether n is in $A_{i,j}$. This is, of course, a finitarily computable test. If one of these tests succeeds, the computer signals to the computer at its parent node. (In Davies's model, each test $A_{i,j}$ will be performed by a different computer. In Hogarth's model, for each i, there is a single computer that tests $A_{i,j}$ for all j.)

B_i: Each node labelled 'B_i' is associated with a computer that tests whether n is in B_i. Since $B_i = \bigcup_j A_{i,j}$, n is in this set iff the computer received a signal from one of its children. If so, it sends a signal to its parent, indicating that the test passed.

$\overline{B_i}$: Each node labelled '$\overline{B_i}$' is associated with a computer that tests whether n is in $\overline{B_i}$. Since $n \in \overline{B_i}$ iff $n \notin B_i$, this computer signals to its parent iff it did not receive a signal from its child.

C: Finally, the top node (the root) is associated with the test whether n is in C. Since $C = \bigcup_i \overline{B_i}$, n is in this set iff the computer at this node received a signal from one of its children.

We thus have a sound and complete test for membership of arbitrary Σ_2 sets. To test arbitrary Π_2 sets, we simply add a single node, labelled '$\overline{C}$', to the top of the tree.

This tree-based treatment of SAD computation follows (Welch [2008], pp. 664–7), and Welch has generalized it as follows. Finite path trees for SAD computers may be constructed using the following rules, and any tree that cannot be constructed using these rules is not the tree for any SAD computer:

(1) Terminal nodes (i.e., nodes with no children) are associated with finitarily computable sets, that is, Π_0 or Σ_0 sets.

(2) A node associated with a Σ_α set A (with $\alpha \geq 1$) has one parent, associated with $\overline{A}$, which is a Π_α set.

(3) Given a sequence of sets $A_1, A_2, \ldots$ such that both:

  (i) $(\forall i \in \mathbb{N})(\exists \beta_i)(A_i \text{ is } \Pi_{\beta_i})$; and

  (ii) there is a finitarily computable function, f, such that for every $i \in \mathbb{N}$ the value of $f(i)$ is the code of a program, and this program can be run by a SAD_{β_i} machine to determine whether any natural number is a member of A_i;

  then the nodes associated with these sets have a single parent, associated with $A = \bigcup_i A_i$. In this case, A is Σ_α, where α is the least ordinal greater than every ordinal β_i.

Condition (1) is justified by the fact that a finitary computer can reliably determine membership of all and only the finitarily computable sets. Condition (2) is justified by the fact that the complement of a Σ_α set is a Π_α set, and that a finitary computer can, trivially, infer whether n is in $\overline{A}$ from knowing whether n is in A (and hence signal upwards iff it receives no signal). Condition (3) captures the idea of standing 'after' an ω-sequence of tests. (For example, when testing the Σ_2 set C above, we aggregated the result of infinitely many Π_1 queries.)

By induction, to determine whether n is in a Π_m set, we require a tree whose branches have no more than $2m + 1$ nodes (Welch [2008], p. 666).

Consequently, given any *arithmetical* set, we can describe a setup that would be able to determine membership of that set (Hogarth [1994], pp. 128–32, [2004], pp. 684–9; Welch [2008], p. 666).

By considering SAD computers corresponding to trees with no fixed finite upper bound on the length of their branches, we can determine membership of Σ_α or Π_α sets, for transfinite ordinals α. However, condition (3) entails that the entire tree must be (finitarily) recursively describable, and hence that α must be less than the least non-recursive ordinal, ω_1^{CK}. Thus, the extent of SAD computability is the class of the *hyperarithmetical* sets (Welch [2008], p. 667).[5]

2 Physical Computability

This completes my explanation of SAD computers. For the remainder of this paper, I shall investigate whether SAD computation should affect our views about which functions are computable.

[5] The relationship between finite path trees and the (finitarily) recursive ordinals is discussed in (Rogers [1967], pp. 392–7). For a general treatment of the hyperarithmetical hierarchy and hypercomputation, see (Rogers [1967], Chapter 16).

This question can be understood in two ways, which should be kept firmly distinct. In Section 3, I consider the question in terms of effective computability. In this section, I consider the question in terms of physical computability. For convenience, in this section I confine my attention to SAD_1 computers: they are physically possible if any SAD computers are, and they raise serious questions about physical computability.

2.1 The Physical Church–Turing Thesis

I treat the notion of physical computability as follows:

Physical Computability: A function, f, is physically computable at a world if and only if there is a machine 'blueprint' such that, for all x, it is physically possible for there to be a machine instantiating that blueprint which physically outputs $f(x)$ on input x.

Understood in this way, the answer to the question 'which functions are physically computable?' obviously depends upon which notion of physical possibility is in play. I will discuss the appropriate sense of physical possibility in some detail below.

The Physical Church–Turing Thesis states that the physically computable functions are exactly the Turing computable functions. So, if there is some sense in which it is possible to build a SAD computer, the Physical Church–Turing Thesis is false in that sense.[6]

I have no desire to show that the Physical Church–Turing Thesis is true, nor to determine what the physically computable functions are. I simply want to show how the Physical Church–Turing Thesis will have to be defended, given that we have described physical models of SAD computability. My discussion comes in two halves. In Section 2.2, I consider deterministic barriers to physical computations. In Section 2.3, I consider probabilistic barriers to physical computations.

2.2 Deterministic barriers to physical computation

Suppose we are using a SAD computer to test whether n is in some Σ_1 set $\{n : (\exists x)\phi(n, x)\}$. The machine must be able to test, for every x, whether $\phi(n, x)$. But there may be *deterministic* physical barriers that prevent it from doing so.[7] For example, note that the computer's memory must be allowed to grow without

[6] Earman ([1995], p. 120) notes that there is a way of reading the Physical Church–Turing Thesis in which it is untouched by such considerations. I discuss this in Section 3.4, footnote 22.

[7] A *deterministic* barrier is a non-probabilistic barrier that is deductively entailed by the physical laws of the world, plus the initial conditions for those laws, plus any further physical constraints in that world (e.g., conditions that prevent the existence of certain kinds of singularity; see Earman [1995], Chapter 3).

limit, even if only to store the increasing values of x. Consequently, the mass of the computer's memory must be permitted to grow without limit.[8] But if there is a deterministic finite upper bound on the amount of mass/energy in the universe, then there is a clear sense in which this is not possible. In which case SAD computation will be physically impossible in this sense. (Again, whether this is the *right* sense of physical possibility will be discussed below.)

Before proceeding, I should note that there is no particular reason for us to focus on mass/energy. It is merely a simple, illustrative example of a deterministic barrier that would prohibit physical SAD computation. Earman and Norton ([1993], pp. 30–40) have raised several rather more sophisticated barriers to the physical realization of one of Hogarth's SAD computers. Etesi, Németi, and Dávid have addressed many of these barriers by locating their SAD computers in the Kerr spacetime, which is a special kind of Malament–Hogarth spacetime (Etesi and Németi [2002]; Németi and Dávid [2006]). I shall not attempt to argue that there *are* any deterministic physical barriers to physical computation; if there are none, then the Physical Church–Turing Thesis is false (pending the discussion in Section 2.3). I just want to consider how the Physical Church–Turing Thesis stands, *given* that there are such barriers. So, for convenience, I shall continue to discuss the toy example of a barrier imposed by mass/energy.

In the presence of this barrier, SAD computers are physically impossible. But one might think that, even in this situation, SAD computation poses an interesting philosophical challenge to the Physical Church–Turing Thesis. In an early paper, Hogarth wrote:

> You don't want to rubbish a hypothetical computer—Turing or non-Turing—simply because it can't fit into our universe. If you do, you'll leave your precious Turing machine to the mercy of the cosmologists. (Hogarth [1994], pp. 136–7)[9]

In a similar vein, Davies writes:

> [T]he rejection of [SAD] machines on the grounds of physical implausibility should imply the same attitude towards Turing machines. (Davies [2001], p. 679)

I suspect that the following argument is at work. Suppose we have a physical Turing machine, programed with some algorithm for testing whether a natural number n is in some set S. The amount of tape and energy required to test

[8] Assuming that matter is not infinitely divisible. The fact that matter is not infinitely divisible imposes a similar deterministic barrier to the physical possibility of Davies's SAD machines.

[9] He has since distanced himself from this line of argument, favouring the arguments discussed in Section 3 (Hogarth [1996], p. 116; [2004], [2009]).

whether n is in S will grow without limit as we give our machine larger and larger values of n. So if there is some deterministic finite upper bound on the amount of mass/energy in the universe, then there will be some natural number n such that our machine cannot physically compute whether n is in S, even though S is Turing computable.[10] This would falsify the Physical Church–Turing Thesis. In order to keep the Thesis alive, we must relax the notion of 'physical possibility' in question. We must be permitted to abstract from certain upper bounds to physical computability, such as those imposed by limited mass/energy. But if it is legitimate for advocates of Turing computability to relax the notion of 'physical possibility' in question, then advocates of SAD computability should be allowed to do exactly the same. In which case (the argument runs) the SAD computable functions are physically computable in exactly the same sense that the Turing computable functions are physically computable.

This argument raises a dilemma against proponents of the Physical Church–Turing Thesis. Either the notion of physical possibility is too strict to allow unrestricted physical Turing computation, or it is too weak to prevent physical SAD computation. I contend that the dilemma is spurious, since physical Turing machines and physical SAD machines require different notions of physical possibility. Consequently, there is room to defend the Physical Church–Turing Thesis.

The most obvious objection to the dilemma is that there may be deterministic barriers to SAD computation that are not barriers to unrestricted Turing computation. For example, if matter is not infinitely divisible, we cannot build Davies's SAD computers.[11] Likewise, whilst Malament–Hogarth spacetimes are consistent with the equations of general relativity, they violate strong cosmic censorship, 'which states that naked singularities do not develop in physically reasonable models of general relativity theory' (Earman and Norton [1993], p. 35; also see Earman [1995], Chapter 3). So if strong cosmic censorship holds, then it is physically impossible to build one of Hogarth's SAD computers. But neither the finite divisibility of matter, nor strong cosmic censorship, would prevent us from constructing physical Turing machines that can run for arbitrary finite periods of time. In short: there may be a perfectly good, natural sense of physical possibility according to which the physically computable functions are exactly the Turing computable functions.

A more interesting objection to the dilemma arises when we consider deterministic barriers that affect physical Turing and physical SAD computers alike,

[10] Of course, a different physical setup—perhaps an Abacus machine rather than a Turing machine, running a better algorithm, and using better hardware—might be able to determine whether n is in S within these limits. But there will be some finite $n' > n$ such that this setup cannot determine whether n' is in S.

[11] Even if matter were infinitely divisible, it might not be possible to make miniature computers 'all the way down'. For example, there might be an ineliminable level of background thermal noise that deterministically disrupts machines beneath a certain mass.

such as a finite limit on mass/energy. Even here, there is an important difference in kind between Turing computers and SAD computers. Suppose that, for each physically possible world, there is some finite upper limit m on the mass/energy in that world ($\forall w \exists m Lmw$). Then, for reasons stated earlier, there is no world in which a SAD computation can be performed. But we cannot infer from this that there is some upper limit m such that, for every world, the amount of mass/energy in that world is less than m ($\exists m \forall w Lmw$). That would just be a quantifier–shift fallacy. Indeed, it is consistent with this situation to suppose that, for any finite value of m, there is a world with m matter/energy. In which case, for any Turing computable set S, any input n, and any fixed physical realization of a mathematical algorithm for computing membership of S, there is a world containing a physical realization of a Turing machine that computes whether n is in S.[12] That is: every Turing computable function will be physically computable.

Accordingly, even if there is an actual deterministic upper barrier to Turing computability, the Physical Church–Turing Thesis may be defensible. We just need to find a legitimate sense of physical possibility such that:

(i) for every physically possible world, there is some deterministic upper barrier to finitary physical computation within that world;

and

(ii) for every deterministic upper barrier to finitary physical computation, there is a world that transcends that barrier.

The first condition prevents SAD computation in every world; the second makes every Turing computation physically possible. We can illustrate these conditions in terms of mass/energy as follows. Suppose (i′) that for any initial conditions, the physical laws of the actual world combined with those initial conditions entail that the quantity of matter/energy is finite; and (ii′) that for any finite quantity m of matter/energy, there are possible initial conditions which, together with the physical laws of the actual world, entail the existence of m matter/energy. Both (i) and (ii) are now satisfied, by treating the physically possible worlds as exactly those worlds that have our physical laws, but that may differ in their initial conditions.

To reiterate: I have used a toy example to illustrate a plausible way in which one *could* defend the Physical Church–Turing Thesis. To *actually* defend the Physical Church–Turing Thesis in this way, we would need to show that there is some genuine, natural, non-*ad hoc* notion of physical possibility that satisfies both (i) and (ii). So, this subsection simply points the way to further work that

[12] cf. Cotogno's ([2003], p. 189) remarks on the actual infinity required by a hypercomputer. For commentary on Cotogno's paper, see (Welch [2004]).

can be done to defend the Physical Church–Turing Thesis against the attack from SAD computation on deterministic grounds.

2.3 Probabilistic barriers to physical computation

I now wish to consider probabilistic barriers to physical computation. In particular, I shall consider the barrier posed by the probability that a machine malfunctions during the course of running a computation.[13] My aim is to show that, even if there is no legitimate notion of physical possibility that satisfies both (i) and (ii), the Physical Church–Turing Thesis can be defended on probabilistic grounds.

A machine may malfunction for many reasons: because there is a defect in the physical material that comprises the machine; because of some intrinsically unavoidable probabilistic event somewhere within the machine's hardware; because of interference by background noise; because it collides with another object; etc. As mentioned above, Hogarth's machines must be capable of unlimited growth. But as a machine grows in size, so (naïvely) does the likelihood that it will malfunction for any of the reasons just mentioned. It therefore seems that, in the infinite limit, the probability that one of Hogarth's machines has malfunctioned is 1. But since SAD computation depends essentially upon the machine's behaviour in the infinite limit, it seems that physical SAD computers are essentially useless.

In slightly more detail, let s_n be the probability that a machine successfully executes the *n*th stage of a computation. The probability of success in the infinite limit is the infinite product $\prod_{n=1}^{\omega} s_n$. If the sequence $\langle s_n \rangle$ does not converge to 1, then this infinite product will be 0. Hogarth's machines must add *new* material at each stage. So the probability of a defect at each stage is to some extent independent of the probability of a defect at any previous stage. Indeed, the probability of failure due to intrinsically probabilistic events somewhere within the machine's memory will plausibly *increase* with the number of stages, since the memory must increase with each stage. In which case, the sequence $\langle s_n \rangle$ will certainly not converge to 1, and the probability that one of Hogarth's SAD machines will malfunction is therefore 1.

Davies believes that this argument may not apply to his model of SAD computation, for he believes that $\langle s_n \rangle$ may converge to 1:

> If the failure is due to impurities in the raw material then the successful operation of the first *n* machines may indicate that the particular sample of raw material used is very pure and hence may lead to a very much higher probability that the later machines will also work as intended. (Davies [2001], p. 678)

[13] For the purposes of the argument, it seems not to matter whether we think of probabilities as objective or subjective.

The difference is that, whereas Hogarth's machines must add new material during the computation, Davies's machines are constructed from a fixed starting 'lump'. Nonetheless, I doubt that Davies's claim is warranted. Presumably, miniscule impurities in the material will only affect very small machines; but then the successful operation of the first n machines is expected whether the material is pure or impure. As before, the probability of failure due to impurity therefore increases at each stage. Furthermore, there are other potential sources of failure to consider. For example, on Davies's model, the probability of failure due to thermal noise must surely be larger at each stage than it was at any previous stage, since smaller computers will be more likely to be significantly disrupted by very minor levels of background noise. So the probability of failure due to thermal noise, and hence the probability of failure *simpliciter*, is 1 in the limit.[14]

SAD computers would therefore be useless, for probabilistic reasons. But we should ask whether these probabilistic arguments generalize to physical Turing machines.[15] As we noted earlier, there are Turing computations that will take any actual physical Turing machine an extremely long time to perform, and that will require an enormous tape. In such cases, the likelihood that the Turing machine will malfunction during the course of the computation is extremely close to 1. This would seem to compromise the Physical Church–Turing Thesis.

There is a difference in kind here. Turing computers may be unlikely to succeed; SAD computers have probability 0 of succeeding. In certain situations, Turing computers are unlikely to be useful, but might be made more useful by repeating the computation many times over; SAD computers are genuinely useless, and cannot be made useful.[16]

This difference in kind can also be illustrated by returning to an idea expressed in the previous subsection. Given a fixed algorithm for computing whether any number is in a Turing computable set S, a defender of the Physical Church–Turing Thesis may be able to establish the following claim. For every natural number n, there is a world in which there is an acceptable probability that a physical Turing machine successfully tests whether n is in S. In contrast, in *every* world, the probability that a SAD machine malfunctions when testing whether n is in some Σ_1 set is 1 (if n is not in that set). Accordingly, the Physical

[14] Davies ([2001], p. 678) refuses to engage with the problem posed by thermal noise, on the grounds that our current best theory of thermal noise rests on the assumption that matter is not infinitely divisible. But this simply highlights that Davies's model is physically impossible for deterministic reasons, and that these deterministic reasons do not render Turing machines physically impossible (see Section 2.2).

[15] Thanks to an anonymous referee for raising this question.

[16] Of course, to have probability 0 of succeeding is not the same as necessarily failing. There is a possible state of affairs in which the SAD computer succeeds, even though the probability of obtaining that state of affairs is 0.

Church–Turing Thesis could be defended on probabilistic grounds, if we could claim that:

(i*) for every physically possible world, the probability that a machine successfully completes an actual ω-sequence of steps in a physical computation is 0;

and

(ii*) for any physical computation consisting of a finite number of steps, there is a physically possible world in which a machine has a probability greater than 0 of successfully completing that computation.

As before, the first condition makes SAD computers useless in every world; the second leaves Turing computers useful.

To conclude this section: I have suggested there may be a sense of physical possibility in which all and only the physically computable functions are Turing computable. This does not show that the Physical Church–Turing Thesis is true; rather, it shows where further debates concerning the Physical Church–Turing Thesis should focus. Moreover, it shows that the Physical Church–Turing Thesis is not refuted by the very *idea* of SAD computation.

3 Effective Computability

In this final section, I wish to consider whether thinking about SAD computability should change our views about *effective* computability.

Hogarth has argued that considering SAD computability reveals a deep analogy between geometry and the theory of computation. He maintains that this analogy shows that we should simply abandon the notion of *effectiveness*.

I will argue that Hogarth's attack on effective computability is a non-sequitur, and will contest the depth of his analogy between geometry and computability. My claim is that SAD computability casts no shadow over the notion of finitary effective computability, and so has no consequences for the Effective Church–Turing Thesis.

3.1 The Effective Church–Turing Thesis

The notion of effective computability requires some explication, since it is very different from the notion of physical computability.[17]

[17] I am here restricting my attention to finitary algorithms; in Section 3.4 I will consider 'SAD-algorithms'. I follow Smith ([2007], pp. 315–23) in linking effective computability to the semi-formal notion of an algorithm. Be warned that some authors differ: for example, Etesi and Németi ([2002], p. 348) use 'effective computability' to mean 'physical computability'.

Effective computability: A function, f, is effectively computable if and only if there is an algorithm that takes any input argument x to $f(x)$.

An *algorithm* is a finitely describable, step-by-step process, followed 'dumbly', which delivers any output in finitely many steps. This is a semi-formal concept, but the concept is relatively precise in its application, since relevant idealizations have already been made.[18]

One such idealization is the length of the computational procedure. So long as any output from an algorithm is delivered in a finite number of steps (on each input), it is irrelevant how many steps it takes (on each input).

Another idealization is atemporality. A mathematics teacher might legitimately use temporal vocabulary while proving that Euclid's algorithm is guaranteed to terminate 'eventually', but we should treat this as an heuristic gloss. Her point is simply that, applied to any pair of numbers, there is always an m such that the output is constant for $n \geq m$ steps in the algorithm. To grasp her proof, the students require the notion of a well-ordered sequence, so that they can see the sequence of steps during an application of Euclid's algorithm *as* a sequence. But there is no reason to demand that their notion of a well-ordered sequence must be a *temporal* notion. So, for just this reason, the notion of an algorithm should not be thought of as intrinsically temporal.

The notion of an effective procedure, or algorithm, is central to mathematics and logic. Mathematicians employed many algorithms long before Church's and Turing's work (see Chabert [1999]), and they continue to. Here are some examples. The syntax of a (finitary) formal language is defined recursively, so there is an effective test to determine whether any given string is a formula. When we show that two theories with different axiomatizations are equivalent, we may do this by offering an effective method for transforming any proof in one theory into a proof in the other. For any putative proof in a formal system, there is an effective procedure for determining whether or not it is indeed a proof. And as a final example: if the Effective Church–Turing Thesis is correct, then Peano Arithmetic tracks the notion of an effective procedure, since the Turing computable functions are exactly those functions that Peano Arithmetic can capture.[19]

It is precisely because the notion of effective computability is central to mathematics and logic that Church, Turing, and everyone else wanted to

[18] Shapiro ([2007]) provides a fascinating discussion of 'open-texture' in the notion of computability. Open-texture seems primarily to affect pre-formal, relatively inchoate concepts; by the time we have reached the semi-formal notion of an *algorithm*, I think that we are relatively free from open-texture. In this spirit, see (Smith [2007], pp. 324–5).

[19] The same goes for Robinson Arithmetic. See, for example, (Smith [2007], pp. 277–8). I follow Smith's ([2007], pp. 35–6) use of 'captures'. A theory T captures the function f iff there is a formula ϕ in the language of T such that, for all m, n: if $f(m) = n$ then $\mathrm{T} \vdash \phi(m, n)$, whereas if $f(m) \neq n$ then $\mathrm{T} \vdash \neg\phi(m, n)$.

characterize the notion of effective computability in the first place. They wished to provide extensionally correct necessary and sufficient conditions for following an algorithm. The Effective Church–Turing Thesis supplies these conditions, by stating that the effectively computable functions are exactly the Turing computable functions.

3.2 Hogarth's challenge to the Effective Church–Turing Thesis

Hogarth dismisses the notion of effective computability altogether. This leads him to maintain that the Effective Church–Turing Thesis is not so much false as *misguided*. He rejects effective computability by analogy with Euclidean and non-Euclidean geometry, and by elucidating the notion of a 'two-sided concept' (Hogarth [2004], pp. 689–90, [2009], pp. 287–8). In this subsection, I shall simply summarize Hogarth's argument and discuss its scope; I shall postpone all criticism of the argument until the next subsection.

On the one hand, there are various formal theories of geometry. There are Euclidean geometries, and there are canonical non-Euclidean geometries, such as Lorentz and Riemannian geometry. These formal theories of geometry are, in a sense, on a par. They are equiconsistent and they 'do not compete' with each other (Hogarth [2004], pp. 689–90), except in the weak sense that one theory may be more interesting for mathematicians to work with than the other.

On the other hand, physics employs some formal theory of geometry to describe the spacetime structure of our physical world. As physical theories, Euclidean and non-Euclidean geometries are in competition with one another: they vie to be *the* geometry that physicists use. Geometry is therefore a 'two-sided concept': it has a formal side and a physical side.

Hogarth maintains that geometry is *only* two-sided. When someone asks whether the ratio between the circumference and the diameter of every circle is π, they might be asking 'is π the ratio for all physical circles?', or they might be asking 'in the formal theory T, is π the ratio for all circles?' Both of these are sensible questions, and we know what it takes to answer them. But suppose our questioner insists that she is not asking about physical or formal circles; she insists that she is instead asking about the circles that are given to us by our 'intuitive concept of a circle', or asking about 'real circles', or some-such. Hogarth maintains that there is nothing we can say to such a person; they need to be re-educated, rather than answered.

Hogarth maintains that the same is true of Turing computability and non-Turing computabilities. There are various formal theories of computability, such as (formal) Turing computability, (formal) Abacus computability, and (formal) SAD computability. Then there is physical computability, which we considered in Section 2. But, for Hogarth, there is no third notion of computability:

[The Effective] Church–Turing Thesis is explained away, for the very question it proposes to answer—'which computational procedure, or computer, captures what is "intuitively computable["]?'—is premised on there being a fundamentally distinguished computational procedure or computer. There is none. (Hogarth [2004], p. 690)

This is how Hogarth's challenge is meant to work. In the next subsection, I shall argue that the challenge is misdirected. Before that, it is worth considering the *scope* of Hogarth's challenge.

If Hogarth is right that the concept of effectiveness is bankrupt, then all supposedly effective procedures will have to be replaced with physical or formal procedures. We can get a taste of what this would mean by reconsidering two examples (from Section 3.1) of the use of effective procedures.

Example 1. Suppose we have proved the equivalence of two theories by supplying an effective method for translating any proof in one theory into a proof in the other. Hogarth will not accept this 'effective' method. He might replace it with a formal procedure in some formal system, but then our translation method only shows that the theories are equivalent *relative* to that formal system; relative to other formal systems, with other formal procedures, they will not be equivalent. So if no formal system is privileged over any other, then it is impossible to show the *absolute* equivalence of any two theories. But, for Hogarth, the only way that a formal system can be privileged is by being *physically* privileged. So, for Hogarth, we can only demonstrate the absolute equivalence of two theories by undertaking some physical demonstration. If our 'effective procedure' depended on induction on the complexity of proofs, say, then we would have to hope that induction on complexity is *physically* legitimate.

Example 2. Suppose we want to know whether some finite sequence of sentences, p, counts as a proof in a finitary formal deductive system, Δ. There is an effective procedure to determine this—we just check whether each step of p follows some rule/axiom of Δ—but Hogarth will not accept this 'effective' procedure. He might replace it with a formal procedure in some formal system, T, but this will only establish that, *relative to* T, p is a proof in Δ; we cannot show this *absolutely*. Alternatively, Hogarth might treat the procedure for inspecting putative proofs as a physical procedure. But then whether p is a proof in Δ or not depends upon whether it is *physically possible* to check the legitimacy of each step in the sequence p. For Hogarth, provability and deducibility turn out to depend upon physics.

I should emphasize that these two examples are not presented as arguments *against* Hogarth's challenge. They simply illustrate that we are playing for high stakes. They show that, if Hogarth's challenge is successful, we shall have to become naturalists (or formalists) about large swathes of logic and mathematics. Whilst this does not constitute an argument against Hogarth's challenge, it does highlight a difference between computability and geometry. I take it that we are willing to jettison the idea that there is a 'privileged' geometry because (with apologies to Kantians) we think that geometry has no central, foundational role in mathematics or logic: we can deny that Euclidean geometry is *a priori*, without becoming widespread naturalists (or formalists). The case of computability seems rather different: the preceding examples showed that abandoning any 'privileged' notion of computability would have revisionary naturalist (or formalist) consequences.

Of course, if there are good *general* arguments for naturalism (or formalism), then we shall have to accept such revisionary naturalism (or formalism) anyway. But completely general arguments for naturalism (or formalism) will render Hogarth's own challenge against effective computability superfluous: if we are convinced of naturalism (or formalism) *in general*, then we shall certainly only be interested in physical (or formal) questions about any *particular* subject area, such as computability. In what follows, therefore, I shall only consider arguments against effectiveness that arise directly from considering SAD computability, and do not presuppose a prior general commitment to naturalism (or formalism).

3.3 Arguing from SAD computability is a non-sequitur

In particular, I shall focus on the following argument, which I think fairly summarizes Hogarth's attack on effectiveness:

(a) Which functions can be computed by a (configuration of) physical Turing machine(s) depends upon the structure of spacetime.

(b) The structure of spacetime is either to be determined by physicists, or to be stipulated within some formal system; there is no third way.

(c) So which functions can be computed by a (configuration of) Turing machine(s) is either a purely formal or purely physical question; there are no interesting questions about effective computability.

I will not contest premise (b). My contention is with premise (a), which begs the question from the outset.

Premise (a) reflects Hogarth's interest in 'painting ordinary Turing machine hardware on a variety of different geometries' ([2009], p. 277). It also reflect's Hogarth's belief that '[p]ure computability is concerned with the logico/mathematical structure of each of the many computers now on offer' ([2004], p. 690). This is to treat formal theories of computability merely as formalizations of some physical theory of computability.

In this context, this simply misses the point of Church's, Turing's, and everyone else's efforts. An attempt to formalize the notion of effective computability is not an attempt to characterize the logical structure of some *physical* computer. It is an attempt to characterize a notion that has a central, foundational role in logic and mathematics. (One might even say that it is an attempt to characterize large parts of the structure of logic and mathematics *itself*.) This role is distinct from any particular physical or formal theory, unless naturalism (or formalism) is the correct approach to logic and mathematics.

In short, unless we *already* think that naturalism (or formalism) is the correct approach to logic and mathematics, we shall think that effectiveness is neither a physical nor a formal concept. As I have explained, if we are already naturalists (or formalists), we shall have no need for Hogarth's argument against effective computability. But if we believe that effectiveness is neither physical nor formal, then painting *physical* realizations of a *formal* theory of computability (a physical Turing machine) onto a strange geometry cannot present any challenge to the non-physical, non-formal idea of an *effective* procedure. Hogarth's argument is simply misdirected.

We can illustrate this point dramatically by modifying Hogarth's setup slightly. Instead of painting physical Turing computers onto a Malament–Hogarth spacetime, we shall paint *effective* computers onto a Malament–Hogarth spacetime. To do this, we can simply replace the machines in Hogarth's model with tribes of people. These people work as 'computers' in Turing's original sense: that is, they dumbly follow a finite set of instructions, step-by-step ([1936], Section 9). As tribe members approach death, younger members of the tribe take on the work of their dying predecessors. The result is an unbroken ω-sequence of calculations on a collective worldline, and a SAD_1 computer. But in this case, we are constructing SAD computability from a core of effective computability, rather than from a core of physical computability: we have an *effective* SAD computer.[20]

Evidently, it is just a non-sequitur to argue from the idea of SAD computability to the claim that the notion of effectiveness is bankrupt. Far from getting rid of the notion of effectiveness, introducing SAD computability just allows us to consider *both* finitary and SAD effective computability. The very

[20] This describes a temporal model of an 'effective SAD computer'. It is worth noting that this does not conflict with the claim, made in Section 3.1, that the notion of a (finitary) algorithm is not intrinsically temporal. We might come to the notion of a (finitary) algorithm by idealizing away the specifically temporal aspects of Turing's description of a (finitary) effective computer. In the same way, having described the intuitive model of an effective SAD computer, we idealize away the temporal aspects of the model, reaching the notion of an atemporal 'SAD algorithm', or an 'effective SAD procedure'.

One might worry that the tribe could be wiped out by a plague; but this is not essentially different from the case where Hogarth's machine is destroyed by collision with a passing asteroid, or where the machine malfunctions due to defects in its components. One might worry that the tribe will not be able to get enough matter/energy to continue computing forever; but the same sort of problems arise with Hogarth's original machine. Any solutions that Hogarth offers to the problems posed in Section 2 can be used here.

idea of an effective procedure—as something neither formal nor physical—is entirely unscathed by thinking about SAD computability.

3.4 SAD computability is built from finitary computability

Nonetheless, SAD computability presents some challenge to the Effective Church–Turing Thesis. If finitary and SAD effective computability are on a par (in some sense), then there is no *single* privileged notion of effective computability. In which case, if the Effective Church–Turing Thesis assumed that there is a single privileged notion of effective computability, then the Thesis would still be fundamentally wrong-headed. This would be a different kind of 'wrong-headedness' from the kind that Hogarth has in mind (Hogarth holds that the very *notion* of effectiveness, whether finitary or infinitary, is bankrupt), but it would still pose serious problems for the Thesis.

In this subsection, I argue that effective finitary computability *is* privileged over effective SAD computability. In doing so, I establish the following *dis*analogy between geometry and computability. In geometry, Euclidean parallel lines are merely special cases of *geodesics*, and they generate no particular interest outside of Euclidean geometry. By contrast, I shall show that finitary computation is absolutely central to SAD computation, and so is *always interesting* in computability theory.[21] This will demonstrate that finitary computability is privileged over SAD computability and, *a fortiori*, finitary effective computability is privileged over SAD effective computability.

Considering Hogarth's model of SAD_1 computation, Earman notes that

> two levels of computation need to be distinguished: the first corresponding to what the [computer on the worldline γ] can do, the second to what [the user] can infer by having access to all of [the computer]'s labors. (Earman [1995], p. 120)[22]

In brief, we do not know what SAD_1 computability amounts to until we know what a finitary computer can achieve. This will be readily conceded by Hogarth, given that his physical model of SAD_1 computation arises by 'painting ordinary Turing machine hardware on a variety of different geometries' ([2009], p. 277).

The relevance of finitary computation to SAD_1 computation does not depend upon the particular details of Hogarth's physical model of SAD_1 computation. As explained in Section 1.1, SAD_1 computers are created simply by allowing

[21] Thanks to an anonymous referee for suggesting I consider Hogarth's challenge in terms of geodesics.

[22] Earman continues by suggesting that the Physical Church–Turing Thesis can be treated as a thesis about what finitary computers can do on their worldline, rather than as a thesis about what can be physically computed *tout court*. Treated in this way, considerations of SAD computability will leave the Physical Church–Turing Thesis untouched. My discussion in Section 2 treats the Thesis as aimed at physical computability *tout court*.

someone to stand 'after' an ω-sequence of computations performed by an essentially *finitary* computer. So finitary computation is central to the very idea of a SAD_1 computer. (At least, as I have presented it; I shall soon discuss whether alternative presentations might be preferred.)

The importance of finitary computation becomes even more obvious when we consider Welch's ([2008]) completely general characterization of the hierarchy of SAD computers, as summarized above in Section 1.5. Condition (1) states that, at base, all the computation that takes place in a SAD_α computer must be built up from finitary computers. Furthermore, condition (3) ultimately entails that the tree-structure of any SAD_α computer must be given by a (finitarily) recursive ordinal; that is, the structure of the computational tree must *itself* be finitarily computable.[23] In short, SAD computation is built up from finitary computation in finitarily computable steps.

This all contrasts sharply with the case of geometry. Euclidean geometry is not built up from Lorentz geometry, and Lorentz geometry is not built up from Euclidean geometry.[24] On the contrary: these two geometric theories have mutually contradictory axioms concerning parallel lines and so have different theorems about circles (for example). By contrast, the axioms of the formal theory of SAD computability will *contain* all of the axioms of the formal theory of Turing computability. So this is the point at which Hogarth's analogy with geometry collapses.

With all of this in mind, let us return to the story of indentured tribes, as told in Section 3.3. If we could send indentured tribes off towards Malament–Hogarth singularities, then the finitarily effectively computable functions might not exhaust the effectively computable functions *simpliciter*. But even in this case, finitary effective computability will remain an extremely important notion. To know whether the *effective* SAD_α computable functions are all and only the *Turing* SAD_α computable functions, it is necessary and sufficient to know whether the finitarily effectively computable functions are all and only the Turing computable functions. Consequently, whether or not we are interested in SAD computability, the answer to the question 'what are the finitarily effectively

[23] To be fair to Hogarth, note that condition (3) is Welch's, not Hogarth's. But I do not think that Hogarth could reject the condition. Welch ([2008], p. 667) argues that Hogarth must adopt condition (3) since, without it, the 'description of this sequence may itself be beyond the computational powers of us or our spacetime-regions', which would render the computer essentially useless to us. To supplement this argument, I note the following. If we impose no constraint on the function f, we quickly reach triviality. *Any* subset of the naturals can be formed by countable union of Π_0 sets, since the union of all the sets that are singletons of members of S is just S, and every singleton set is Π_0. So if we impose no constraints upon f, we can determine membership of any set using a (meagre) SAD_1 setup.

[24] That we can construct *models* of n-dimensional Lorentz geometry in $(n+1)$-dimensional Euclidean geometry is irrelevant here. We are talking about *theories* of computability and geometry, not models of those theories. Furthermore, Lorentz geometry has models in its own right; we do not need to think of it as having Euclidean geometry at its heart, in the way that SAD computability has finitary computability at its heart.

computable functions?' will be of paramount importance. Reading the Effective Church–Turing Thesis in this way, it seems anything but misguided.

Before concluding, it is worth asking whether SAD computability's reliance on finitary computability is an artefact of the way that Hogarth, Welch, and I have presented the theory of SAD computability, or whether it is essential to SAD computability. We have built SAD computers 'from the bottom up', starting with finitarily computable sets, and ultimately constructing the hyperarithmetical hierarchy on top of them. Perhaps this is an unfairly constructivist prejudice, which could be avoided by starting 'from the top' and working downwards. If we can do this, then finitary computability might have a less central role in SAD computability.

But recall from Section 1.2 that we wanted to construct a notion of hypercomputability that did not require any individual component of the hypercomputer to complete a supertask. To demonstrate that no component completes a supertask, we have to consider the behaviour of the individual components. So, even if we start from the top, we will have to consider what happens at the bottom. Considerations of finitary computation will still be central to any theory that looks like SAD computability.

To be sure, we can develop alternative theories of hypercomputability that *do* involve supertasks. In the extreme case, one need only postulate the existence of an oracle that can determine membership of any hyperarithmetical set.[25] But simply stating that 'there could be an oracle', or 'there could be a supertask machine' presents no philosophical challenge for any notion of computability. After all, if God exists, She can compute any function; but Her possible existence does not threaten to collapse the distinction between computable and non-computable functions, or threaten to undermine the Effective Church–Turing Thesis.[26]

I therefore take it that finitary computability must be at the heart of any theory of hypercomputability that could reasonably claim to have any effect on the Effective Church–Turing Thesis. But then my previous comments will go through exactly as before: we shall be right to privilege questions about effective finitary computability. In short, the Effective Church–Turing Thesis seems to address a very significant question in the philosophy of mathematics.

[25] In the less extreme case, Hamkins and Lewis ([2000]) describe a hypercomputer consisting of a Turing machine with an infinitely long tape consisting of an ω-sequence of cells. If, during any ω-sequence of computations, the value printed in a cell converges to a limit l, then the value printed in the cell at the limit stage is l; otherwise, it is some default value. As such, this computer must genuinely perform a supertask (or operate in infinite time). But note that this kind of hypercomputer is formed simply by adding new rules to govern the behaviour of Turing machines in limit stages; that is, it is built up from finitary computability. (Also, note that Hamkins and Lewis do not endorse Hogarth's claim that hypercomputability undermines the very notion of effective computability; they are simply interested in the logical/mathematical structure of these machines.)

[26] Cf. (Benardete [1964], pp. 190–3; Dummett [1991], pp. 347–51).

4 Concluding Remarks

I have aimed to show that there is space to defend the Physical Church–Turing Thesis, and also to raise serious probabilistic concerns about the possibility of physical SAD computation. At the very least, this shows that the very *idea* of SAD computation does not refute the Physical Church–Turing Thesis. But, for all that I have said, SAD computers may one day pose problems for the Physical Church–Turing Thesis (see Section 2).

By contrast, the notion of SAD computation casts no shadow upon the notion of effective computability. How could it, when we can easily construct a theory of effective SAD computability? And when finitary computability (physical, effective, or Turing) is the very kernel around which SAD computability (again, physical, effective, or Turing) is constructed? Hogarth has suggested that we should jettison effective computability, since the idea assumes that there is 'a fundamentally distinguished computational procedure or computer' (Hogarth [2004], p. 690). Indeed, it assumes that there is a fundamentally distinguished *non-physical*, *non-formal* computational procedure, and Hogarth's arguments do not show that this assumption is unreasonable (see Section 3).

Nonetheless, everything I have said so far leaves open the possibility that SAD computability might have some foundational role in logic and mathematics. For example, one might allow that the procedure for checking whether a sequence of sentences constitutes a proof or not should be an effective SAD procedure (rather than an effective finitary procedure). This will give rise to a deviant, infinitary notion of proof.[27] I have shown that effective finitary computability is privileged, and this suggests that the standard (finitary) notion of proof is privileged over the deviant, infinitary notion. But it would require a further argument to show that the deviant notion should be ignored *altogether* as a useful notion of proof. This argument would need to establish that we

[27] cf. Example 2 of Section 3.2. I am greatly indebted to Philip Welch for pointing out that this 'deviant' notion of proof is the admissible fragment of the proof theory for the admissible fragment of the infinitary logic L_{ω_1,ω_0}. A brief explanation is in order. L_{ω_1,ω_0} allows countable infinitary conjunctions and disjunctions, but only finitary quantification (all quantification is first-order). Infinitary disjunction is defined explicitly as $\bigvee \Phi =_{\text{df}} \neg \bigwedge\{\neg\phi : \phi \in \Phi\}$. The proof system for this logic, $\Delta_{\omega_1,\omega_0}$, has all the inference rules and axioms of standard finitary proof systems, plus: (i) a new axiom scheme '$\bigwedge \Phi \to \phi$' for any countable set of formulae Φ and any formula $\phi \in \Phi$, and (ii) a new inference rule '$\phi_0, \phi_1, \ldots, \phi_n, \ldots \vdash \bigwedge_{i\in\mathbb{N}} \phi_i$'. A proof in $\Delta_{\omega_1,\omega_0}$ is any countable sequence of sentences formed according to these rules/axioms (Scott [1965], p. 332; Bell [2006], Section 2). SAD computers could be used to test sequences of formulae to determine whether they are formed by these rules/axioms; but restrictions will apply. Recall that the tree-structure of any SAD computer must correspond to an ordinal $< \omega_1^{\text{CK}}$ (see Section 1.5). Consequently, SAD computers can only handle proofs corresponding to trees with order-type $< \omega_1^{\text{CK}}$. For the same reason, both the new axiom scheme and the inference rule must be restricted: any set of formulae Φ, and any formula ϕ, occurring in a putative proof must be associated with ordinals $< \omega_1^{\text{CK}}$. In sum, we are restricted to the *admissible* fragments of L_{ω_1,ω_0} and $\Delta_{\omega_1,\omega_0}$. See (Barwise [1969]) for details.

should take *no* notice of effective SAD computability. Can such an argument be supplied?

Lingering doubts about supertasks might provide a reason to dismiss the notion of effective SAD computability outright. When I presented the two models of SAD_1 computability, I showed that no single component of either model performs a supertask. But equally, when a SAD_1 computer shows that n is not in some Σ_1 set, no *single* component does this. So if it is correct to say that the procedure demonstrated that n is not in the set, then surely it is correct to say that a supertask was performed. It is just that both the demonstration and the supertask are performed by the plurality (or fusion) of *all* the components in the SAD_1 setup (either the entire chain of machines, in Davies's model, or the machine-plus-user, in Hogarth's model). One might then go on to argue that, since supertasks keep bad company, such as Thomson's Lamp, we should not allow supertasks to infect the idea of effective computability. It would follow that we should ignore the notion of effective SAD computability.

SAD-enthusiasts are likely to respond by claiming that it is a mere finitistic prejudice to tar all supertasks with the same brush:

> [T]he fact that some supertasks are kinematically or dynamically impossible is no more surprising or disturbing than the fact that some ordinary tasks are kinematically or dynamically impossible. (Earman and Norton [1996], p. 235)

In a similar vein, Hogarth ([2009], pp. 287–8) asks us to beware of prejudices inculcated by years of non-exposure to Malament–Hogarth spacetimes.

At this point, an impasse is likely to ensue, between those who regard all supertasks as equally conceptually impossible, and those who do not. I have nothing to add to break this particular impasse. However, I think it may be possible to bypass it.

Certain conceptions of logic and mathematics would give us reason to ignore the notion of effective SAD computability altogether. Consider the Fregean view that the laws of logic and mathematics are 'the laws of thought' (developed in Dummett [1991], p. 1ff). Since effective procedures are central to logic and mathematics (see Section 3.1), it would be natural to treat the notion of an effective procedure as an aspect of the laws of thought. Suppose we also make explicit two further constraints: first, that the laws of thought are the laws of thought *for an individual*; second, that no individual can perform an effective SAD computation (since no individual can have completed an ω-sequence of computations). It will follow that we shall simply have no interest (apart, perhaps, from idle curiosity) in effective SAD computability; *a fortiori*, we will have no interest in the 'deviant' infinitary notion of proof.

To evaluate whether this is a good reason to restrict our attention to effective finitary computability, we will have to answer the general question: What is the status of logic and mathematics? If logic and mathematics should generally be naturalized, for example, then we shall have to abandon Frege's conception of logic and mathematics, and the preceding argument will not go through. But in that case, as I have already discussed, we will have no need for a non-physical, non-formal notion of effective computability. Conversely, the question 'what is the status of logic?' cannot be answered by appeal to SAD computers.

In short, to answer these questions, we shall have to look beyond SAD computability. The philosophical usefulness of the SAD computer has run its course.

Acknowledgements

This paper benefited tremendously from discussion, correspondence, comments, and advice from many people, but most of all: Jeremy Butterfield, Adam Caulton, Mark Hogarth, Luca Incurvati, Michael Potter, Stewart Shapiro, Peter Smith, Philip Welch, and referees for this journal.

Darwin College, Cambridge University
CB3 9EU, UK
button@cantab.net

References

Barwise, J. [1969]: 'Infinitary Logic and Admissible Sets', *Journal of Symbolic Logic*, **34**, pp. 226–52.

Benardete, J. A. [1964]: *Infinity: An Essay in Metaphysics*, Oxford: Clarendon Press.

Bell, J. L. [2006]: 'Infinitary Logic', in E. Zalta (*ed.*), *Stanford Encyclopedia of Philosophy*, <plato.stanford.edu/entries/logic-infinitary/>.

Chabert, J.-L. (*ed.*) [1999]: *A History of Algorithms: From the Pebble to the Microchip*, translated by Chris Weeks, London: Springer.

Cotogno, P. [2003]: 'Hypercomputation and the Physical Church–Turing Thesis', *British Journal for the Philosophy of Science*, **54**, pp. 181–223.

Davies, E. B. [2001]: 'Building Infinite Machines', *British Journal for the Philosophy of Science*, **52**, pp. 671–82.

Dummett, M. [1991]: *The Logical Basis of Metaphysics*, London: Duckworth.

Earman, J. [1995]: *Bangs, Crunches, Whimpers, and Shrieks: Singularities and Acausalities in Relativistic Spacetimes*, Oxford: Oxford University Press.

Earman, J. and Norton, J. D. [1993]: 'Forever is a Day: Supertasks in Pitowsky and Malament–Hogarth Spacetimes', *Philosophy of Science*, **60**, pp. 22–42.

Earman, J. and Norton, J. D. [1996]: 'Infinite Pains: The Trouble with Supertasks', in A. Morton and S. P. Stich (*eds*), *Benacerraf and His Critics*, Oxford: Blackwell. pp. 231–61.

Etesi, G. and Németi, I. [2002]: 'Non-Turing Computations via Malament–Hogarth Space-times', *International Journal of Theoretical Physics*, **41**, pp. 341–70.

Hamkins, J. D. and Lewis, A. [2000]: 'Infinite Time Turing Machines', *Journal of Symbolic Logic*, **65**, pp. 567–604.

Hogarth, M. [1992]: 'Does General Relativity Allow an Observer to View an Eternity in a Finite Time?', *Foundations of Physics Letters*, **5**, pp. 173–81.

Hogarth, M. [1994]: 'Non-Turing Computers and Non-Turing Computability', *PSA: Proceedings of the Biennial Meeting of the Philosophy of Science Association*, 1994, volume 1 (Contributed Papers), pp. 126–38.

Hogarth, M. [1996]: *Predictability, Computability and Spacetime*, Ph.D. Thesis, Cambridge University.

Hogarth, M. [2004]: 'Deciding Arithmetic Using SAD Computers', *British Journal for the Philosophy of Science*, **55**, pp. 681–91.

Hogarth, M. [2009]: 'Non-Turing Computers are the New Non-Euclidean Geometries', *International Journal of Unconventional Computing*, **5**, pp. 277–91.

Németi, I., Dávid, G. [2006]: 'Relativistic Computers and the Turing Barrier', *Journal of Applied Mathematics and Computation*, **178**, pp. 118–42.

Rogers, H. [1967]: *Theory of Recursive Functions and Effective Computability*, Cambridge, MA: MIT Press.

Scott, D. [1965]: 'Logic with Denumerably Long Formulas and Finite Strings of Quantifiers', in J. W. Addison, L. Henkin and A. Tarski (*eds*), *The Theory of Models: Proceedings of the 1963 International Symposium at Berkeley*, Amsterdam: North-Holland Publishing Company, pp. 329–41.

Shapiro, S. [2007]: 'Computability, Proof, and Open-Texture', in A. Olszewski, J. Wolenski and J. Robert (*eds*), *Church's Thesis after 70 Years*, Frankfurt: Ontos Verlag. pp. 420–55.

Smith, P. [2007]: *An Introduction to Gödel's Theorems*, Cambridge: Cambridge University Press.

Thomson, J. F. [1954]: 'Tasks and Supertasks', *Analysis*, **15**, pp. 1–13.

Turing, A. [1936]: 'On Computable Numbers, with an Application to the *Entscheidungsproblem*', *Proceedings of the London Mathematical Society*, **42**, pp. 230–65.

Welch, P. D. [2004]: 'On the Possibility, or Otherwise, of Hypercomputation', *British Journal for the Philosophy of Science*, **55**, pp. 739–46.

Welch, P. D. [2008]: 'The Extent of Computation in Malament–Hogarth Spacetimes', *British Journal for the Philosophy of Science*, **59**, pp. 659–74.