Supertasks do not increase computational power

Oron Shagrir

Abstract: It is generally assumed that supertasks increase computational power. It is argued, for example, that supertask machines can compute beyond the Turing limit, e.g., compute the halting function. We challenge this assumption. We do not deny, however, that supertask machines can compute beyond the Turing limit. Our claim is that the (hyper) computational power of these machines is not related to supertasks, but to the "right kind" of computational structure.

Keywords: Supertasks, Turing machines, accelerating Turing machines, relativistic machines, halting function, hyper computation.

1. Introduction

There are many thought-provoking examples of machines that compute by performing supertasks. Some of these machines, it is argued, are "hyper-computers": they have the ability to compute beyond the Turing limit. Our aim here is to examine the relations between supertasks and computational power. We argue that performing supertasks does not increase computational power. We do *not* deny that there are hyper-computers that perform supertasks. The claim, rather, is that the (hyper) computational power of these machines is not related to the performance of supertasks. Their extra computational power has to do with the fact that these machines have the "right kind" of computational structure.

The term *supertask* refers in the present context to the execution of an infinite number of computation steps in a finite span of time (for a detailed discussion of supertasks, see Laraudogoitia 2009). The term *computational power* refers to the class of functions that can be computed by a device. When saying that "*performing supertasks does not increase computational power*" we mean roughly this: Take some computational structure, say a certain Turing machine, and conjoin it with a certain temporal pattering so that it is now a supertask machine. Think, for example, of the Turing machine (or a realization of it) as performing infinitely many steps in a finite span of time. The claim is that the computational power of this supertask machine is no larger than that of the original structure. The computational power of the accelerating Turing machine does not exceed the power of the "non-accelerating" Turing machine.

The plan of the paper is as follows: Sections 2-4 are about the accelerating Turing machine (ATM), which "is the work-horse of hypercomputation" (Potgieter and Rosinger 2009). When surveying the literature it was discovered that there are two very different notions of ATM. Sections 2 and 3 are about the notion that is defined by Copeland (1998a, 1998b, 2002). According to this definition, the ATM does not include limit stages. We claim that these machines do not compute beyond the Turing limit. Section 4 concerns the other notion of ATM, according to which the machine has a limit stage. These machines, we argue, have the same computational power as similar non-accelerating limit-stage machines, e.g., infinite-time Turing machines. Section 5 is an attempt to generalize the argument to relativistic machines, those that compute in the so-called Malament-Hogarth space-time structures. In Section 6 it is conjectured why we use supertask machines, given that they do not increase computational power.

2. Accelerating Turing machines without a limit stage

What is an accelerating Turing machine? It turns out that the answer is ambiguous. One answer – which we consider first – refers to a *Turing machine* that performs tasks in an accelerated fashion. For example, it completes the first operation in one moment, the second in 1/2 of a moment, the third in 1/4 of a moment, and so on. As such, the machine can complete infinitely many computation steps in a finite span of time, and so can perform *supertasks*. The potential realization of an arbitrary process, in which each step takes half of the time of the previous step, is discussed in Russell (1915), Blake (1926) and Weyl (1949). But it was Ian Stewart who introduced the concept of an accelerating process in the context of a Turing machine: "Imagine a Turing machine with a tape that accelerates so rapidly that it can complete an infinite number of operations in one second" (Stewart 1991:664). Jack Copeland was perhaps the first to coin the term *accelerating Turing machine*: "Imposing the Russell-Blake-Weyl same temporal pattering upon a Turing machine produces an *accelerating* Turing machine" (Copeland 1998a:151).

Copeland does not provide a formal definition, but he does emphasize that the ATM is "a Turing machine pure and simple" (2002:294), and the only difference is the temporal pattering of acceleration. Copeland (1998b) also provides an elegant example of an ATM that arguably computes the halting function. The halting function, it will be recalled, accepts as an argument the pair (m,n) and returns '1' if the m^{th} Turing machine (one can think of m as the code of the machine, or as the index of the machine in some enumeration) halts on input n, and returns '0' otherwise. The halting function is an example of something that cannot be computed by a universal

Turing machine. If the so-called Church-Turing thesis is true, then the halting function is not computable by means of any algorithm whatsoever.

Copeland introduces an accelerating *universal* Turing machine (AUTM) for computing the halting function. AUTM, being a universal machine, can compute everything that can be computed by any other Turing machine. The idea behind universality is wonderfully simple. Since the program of a Turing machine is finite we can encode it into a string of 0's and 1's, and feed the finite code paired with binary data as an input to the universal machine. The universal machine scans the code and simulates the operations of the encoded machine on the given data. Thus when getting a pair of arguments (*m*,*n*), as an input, AUTM simulates the operations of the Turing machine, *m*, as operating on input *n*.

AUTM includes two more ingredients as part of its program. One is an instruction that initializes a designated output cell on the tape with '0', which means "never halted" (we can think of this initialization as the first atomic operation that AUTM performs). Another is a controller, which is a finite set of instructions whose task is to tell, after each simulated operation, whether the simulated machine has halted. If the simulated machine did halt, the controller changes the symbol in the output cell from '0' to '1', which means "halted", and terminates the simulation.

Overall, then, AUTM works as follows:

- It gets as an input a pair of arguments (*m*,*n*), where *m* is the code (or index) of a Turing machine, and *n* is the input to that machine.
- It initializes a designated output cell on the tape with '0', which means "never halted".
- It simulates the operations of the machine, *m*, operating on input *n*, step-by-step.
- After each simulated operation, it examines whether the simulated machine, *m*, has halted.
- If the simulated machine did halt, it changes the symbol in the output cell from '0' to '1', which means "halted", and terminates the simulation.

It is important to note that, thus far, the specified machine does *not* compute the halting function. The specified machine returns '1' if the simulated machine, *m*, halted, and returns no value otherwise. If the simulated machine, *m*, never halts, the specified simulating machine keeps working forever. It never arrives at a halting (end) state, for it keeps simulating the operations of *m*.

However, AUTM is an *accelerating* machine: it performs a supertask. It takes one moment to perform the first atomic operation, half a moment to perform the second operation, a quarter of a moment for the third operation, and so on. If the simulated machine never halts, it takes AUTM 1 + 1/2 + 1/4 + 1/8 +… moments to simulate the entire operations of the encoded machine. It will thus take AUTM no more than two moments to complete the entire simulation. After at most two moments of time, the output cell will show the halting state of the simulated machine: it will show '1' if the simulated machine has halted and '0' otherwise. Thus, Copeland argues, AUTM computes the halting function.

3. AUTM does not compute the halting function

As Copeland notes, the claim that AUTM computes the halting function is seemingly paradoxical: "there is, of course, an air of paradox about this" (Copeland 2002: 290; see also Svozil 1998). On the one hand, an accelerating Turing machine *is* “a Turing machine fair and square” (p. 295). On the other, AUTM computes something – e.g., the halting function – which, provably, is not Turing machine computable. How can we resolve the paradox? Fraser and Akl (2008) deny that an *accelerating* machine is really a Turing machine fair and square: "The accelerating machine cannot be considered a Turing machine, since the Turing machines operate on discrete time

intervals" (p. 84). Our strategy is different. We agree that AUTM is a Turing machine. Our claim is that AUTM does not compute the halting function. That AUTM accelerates, and performs a supertask, makes no computational difference.

Why not? The reason is this: According to the textbook definition, a Turing machine computes the (defined) value of the function *f* for some argument, *x*, just in case the machine reaches an end-stage at which the value of the function *f*(*x*) is printed in a designated output cell (the printing operation itself can take place at some previous stage – the important point is that the value of the function appears in the designated output cell at the end-stage). The end-stage might consist of a specific halt state (Lewis and Papadimitriou 1981:172), but there are other means to indicate that the machine has halted (Boolos and Jeffrey 1980: 22-23; Turing 1936). The *raison d'être* of the end-stage requirement is apparent: If a machine does not reach an end-stage, the value in the designated output cell can be altered at some future stage of the computation; and so there is no hope to define, properly, when the machine computes the value of the function. Assuming this, we can say that AUTM computes the halting function just in case the following conditions are satisfied: (i) AUTM reaches an end-stage at which the value '1' is printed in the designated output cell if the machine *m* halts on input *n*; and (ii) it reaches an end-stage at which the value '0' is printed in the designated output cell if the machine *m* does not halt on input *n*.

But AUTM does not satisfy the second condition. The description of AUTM specifies what would be the configuration of the machine after one moment, then after a one and a half of a moment, then one and three quarters of a moment, and, in general, after $2 - (½)^{k-1}$ moments of time (whereas *k* is the k^{th} computation step). It specifies what would be the configuration of the machine in every moment in the semi-open time segment [0,2). Thus when the simulated machine never halts, no end-

stage of AUTM is forthcoming. No stage prior to the second moment is a good candidate for being an end-stage. Each stage that precedes the second moment is followed by infinitely many other stages of machine computation.

The upshot is that AUTM is doing no better than its non-accelerating cousin, which, we saw, does not compute the halting function. When the simulated machine halts, AUTM returns the value '1', but when the simulated machine never halts, AUTM does not have an end-stage and returns no value. Acceleration and supertasks make no computational difference.

One may argue that there is an end-stage, which is at the second moment. After all, it can be argued, the halting state of the simulated machine, be it '0' or '1', appears in the designated output cell after two moments of time. There are two problems with this suggestion, however. One is that the configuration of the machine, and thus the values in the tape in the second moment, is not part of the specification of AUTM, which applies only for the time segment [0,2). As Benaceraff (1965) has taught us, the specification of the machine's operations, in the time segment [0,2), implies nothing about the value in the output cell after two moments. Whatever appears in the output cell after two moments, be it '0', '1' or gibberish, is just consistent with the AUTM specification (otherwise, you get Thomson-like paradoxes: for the question then becomes what is printed after two moments in the output cell of ATM that computes the partial sums of the infinite series 1,-1,1,-1,… ?; for further discussion see Shagrir 2004).

Moreover, the configuration of the machine at the second moment *cannot* be part of the specified AUTM. To qualify as a state of a *Turing machine*, the configuration of each stage, α+1 – namely, the program's condition (*m*-configurations) q_n, the symbols on the tape, and the location of the read/write head –

is completely determined by the configuration of the previous stage, α. As Turing put it, the "possible behavior of the machine at each moment is determined by the *m*-configurations q_n and the scanned symbol $\xi(r)$" (1936:117). But the value in the output cell after two moments of time cannot be determined by the configuration of *the* previous stage. This is because there is no stage, α, that precedes a second moment stage, $\alpha+1$. There are infinitely many stages between any given stage that precedes the second moment and the configuration of the machine after two moments. Thus the state of the output cell, after two moments of time, cannot be part of a Turing machine.

One may object to the assumption that there must be an end-stage at all. Copeland, for one, argues that it is enough that "the machine can produce values from arguments (for all arguments in the domain), displaying each value at a designated location some pre-specified number of moments after the corresponding argument is presented. The machine may or may not halt once a value has been displayed" (2002:296); this is what he labels “external computation”. We discuss the issue of external computation elsewhere (Copeland and Shagrir, forthcoming). Here it will just be noted that we do not object to the idea of external computation. The difficulty is in defining “the pre-specified number of moments”. It cannot be a number short of two moments because the machine can always alter the value of the output immediately after this number. And it is not trivial to point to the second-moment stage, since this stage is not part of the specified Turing machine (at least when the term *Turing machine* refers to the abstract object studied in theoretical computer science; for a detailed discussion see Copeland and Shagrir, forthcoming).

In closing this section, let us make clear what we do *not* claim. First, we do not dispute the claim that AUTM performs a supertask, namely, completes infinitely

many steps in a finite span of time. It certainly does: if the simulated machine never halts, AUTM completes the simulation, which consists of an infinite number of operations in just two moments. The claim is, rather, that the (super)task that AUTM performs is exactly the task performed by its non-accelerating counterpart. It returns '1' if the simulated machine *m* halts and returns no value if the machine *m* never halts. AUTM just performs the very same task in a finite span of time.

Second, our claim is not about the physical feasibility of supertasks. It is doubtful that AUTM itself is physically feasible (for discussion see Steinhart 2003 and Fearnley 2009). But we can assume for the purpose of the argument that the tape somehow survives the acceleration, even if the rest of machine is gone (Copeland 2002:289). The output cell might show '0', '1' or '17'; if we think of the machine as a physical one, this is dictated by the physical properties of the system and by the laws of nature. Our only claim is that whatever is on the tape after two minutes is not part of the description of AUTM.

Third, we do not challenge the concept of hypercomputation, namely, of machines that compute functions that are not Turing computable. On the contrary, in what follows we discuss other supertask machines that do compute the halting function. The only claim is that AUTM itself is not a hyper-computer, and it does not compute the halting function.

4. Accelerating Turing machines with a limit stage

Interestingly, there are others who define accelerating Turing machines to include a second moment limit stage. Steinhart, for example, writes that "An ATM is a CTM [Classical Turing Machine] with an adjoined limit state" (2002: 272). On this view,

the adjoined limit state is some configuration of the machine as an end-stage, which is an essential part of the concept of an accelerating Turing machine. Steinhart goes on to specify possible definitions of an appropriate limit stage. Other precise definitions have recently been provided in Calude and Staiger (2009), Potgieter and Rosinger (2009), and Svozil (2009). The idea in general is of an infinite-time Turing machine where the value in the designated output cell is a limit of the previous values that this cell has displayed (Hamkins and Lewis 2000, and Hamkins 2002). Thus the machine might work as follows. At the semi-open time segment [0,2) it is just the AUTM specified above, namely, it operates in a classical manner in the sense that "the classical procedure determines the configuration of the machine… at any stage $\alpha + 1$, given the configuration at any stage α" (Hamkins 2002:526). At the second moment, however, "the machine is placed in the special *limit* state, just another of the finitely many states; and the values in the cells of the tapes are updated by computing a kind of limit of the previous values that cell has displayed" (ibid.). Let us call this machine $AUTM^+$, to distinguish it from AUTM.

$AUTM^+$ differs from AUTM in two important respects. First, $AUTM^+$ can compute the halting function. We can easily set it up such that when halting after two moments, it exhibits, at its output cell, the halting state of the simulated Turing machine. The value at the second moment is '1' just in case the machine altered, at some point before the second moment, the value of the output cell from '0' and '1', and then halted. The value at the second moment is '0' otherwise; this value can be seen as the limit of the series of the values that were written in the cell at the stages that preceded the second moment.

Second, $AUTM^+$ is *not a Turing machine*. It no longer has the *computational structure* of a Turing machine. The reason it is not a Turing machine was mentioned

before. One of the machine's stages, $\alpha+1$, is not completely determined by the configuration of the previous stage, α. Its second moment limit state, $\alpha+1$, is not determined by "the classical procedure", given the configuration at stage α. This is simply because there is no such *the* preceding stage α. There are infinitely many stages between any given stage that precedes the second moment and the second moment limit stage. Thus the second moment limit stage is not and cannot be a stage of a Turing machine.

As noted, $AUTM^+$ is a hyper-computer, yet its extra computational power has little to do with acceleration and supertasks. After all, there are *non-accelerating* machines with an adjoined limit state that have exactly the same (hyper) computational power as our $AUTM^+$. Infinite-time Turing machines can compute the halting function. They have the same computational structure as the accelerating machines, but they do not use acceleration or supertasks; they operate in transfinite time. And, of course, we can define other non-accelerating hyper-machines, which share the same computational structure without mentioning time in any essential way (Cohen and Gold, 1978). All this indicates that the extra (hyper) computational power of $AUTM^+$ stems from computational structure *and not from acceleration*. $AUTM^+$ can compute "the uncomputable" simply because its computational structure includes a limit stage (in the final section we discuss the objection that transfinite time machines are computers at all).

5. The argument extended: Relativistic machines

We have considered two setups of supertask machines, namely, the two kinds of accelerating Turing machines. It is now time to examine the extent to which the

argument applies to other setups of supertask machines. There are at least three kinds of additional setups: accelerating machines with a hooter (Copeland 2002), shrinking machines (Davies 2001, Beggs and Tucker 2006, Schaller and Svozil 2009), and relativistic machines (Pitowsky 1990, Hogarth 1992). When computing the halting function, the three setups share some essential features. They all involve a universal machine (or several machines) that simulates the operations of the m^{th} Turing machine operating on input n. We can think of this simulating machine as a Turing machine, or a realization of it. In this respect, the setups do not differ from the accelerating machines discussed earlier. The three setups, however, are also equipped with another component, *separate* from the simulating machine(s), whose main role is to detect a signal from the simulating machine(s). The meaning of this signal is that the simulated machine halts on the input n. We shall focus on the relativistic version of this setup, leaving it to the reader to apply the argument to the other two setups.

Relativistic machines are constructed in the context of certain space-time structures in General Relativity known as Malament-Hogarth structures; e.g., anti de Sitter space-times. In essence these devices rest on the observation that there are solutions to Einstein's equations according to which there are space-times with the following property. The space-time includes a future endless curve γ with a past endpoint q, and it also includes a point p, such that the entire stretch of γ is included in the chronological past of p. This property, it is argued, enables all sorts of infinite computations in a finite span of time, ones that cannot be achieved by a standard Turing machine. Pitowsky (1990) discusses the possibility of deciding open mathematical problems such as Goldbach's conjecture. Hogarth (1994:127-133) points out the non-recursive computational powers of such devices. He also notes that which functions are computable depends on the properties of the space-time. More

recently, Etesi and Németi (2002), Hogarth (2004), Welch (2008) and others further explore the computational powers of these structures, within and beyond the arithmetical hierarchy. We discuss here a variant of the setup suggested in Shagrir and Pitowsky (2003) for computing the halting function; but the setup is not essentially different from the ones suggested by others.

Our machine, *RM* (for relativistic machine), consists of a pair of Turing machines, or of their "physical implementations". The two Turing machines, T_A and T_B, are in communication. T_B moves along γ, and T_A along a future-directed curve that connects the beginning point q of γ with p. The time it takes T_A to travel from q to p is finite, while during that period T_B completes the infinite time trip along γ. This physical setup permits the computation of the halting function. One feeds T_A with input (m,n). T_A prints '0' in its designated output cell, then sends a signal with the pertinent input to T_B. T_B is a universal machine that mimics the computation of the m^{th} Turing machine operating on input n. In other words, T_B calculates the Turing-computable function $f(m,n)$ that returns the output of the m^{th} Turing machine (operating on input n), if this Turing machine halts, and returns no value if this Turing machine does not halt. If T_B halts, it immediately sends a signal back to T_A; if T_B never halts, it never sends a signal. Meanwhile T_A 'waits' during the time it takes T_A to travel from q to p (say, two minutes). If T_A has received a signal from T_B it prints '1', replacing the '0', in the designated output cell. One way or the other, the output cell shows the value of the halting function after two moments of time (of T_A). It is '1' if the m^{th} machine halts on input n, and it is '0'otherwise.

This construction is considered to invoke a supertask, since our *RM*, which consists of the two communicating Turing machines, performs infinitely many computation steps in a finite span of time; the infinitely many steps (when needed)

consist of those that are executed by T_B, and the finite span of time is considered here from the point of view of T_A. This sort of supertask is known as a bifurcated supertask (Laraudogoitia 2009). That our machine consist of *two* communicating Turing machines is not essential. We could replace T_A by an observer, a signal detector, or the like. There are other features that are essential, however. One is that the machine that computes the halting function must have an additional component to T_B, one that can detect the halted-signal sent from T_B. The other essential feature is that this component be located in the finite-time trajectory from q to p. The supertask is defined with respect to this finite-time of the additional component.

The construction of *RM*, in terms of the two communicating machines, is helpful when discussing the relations between supertasks and computational power. At first glance, it seems that performing supertasks increase computational power. *RM*, one might argue, is just a pair of communicating Turing machines. A pair of communicating Turing machines, *under standard time scales*, computes no more than what is computable by a single Turing machine (see, e.g., Gandy 1980). In particular, such a pair, when no supertask is involved, cannot compute the halting function. Still, our *RM* does compute the halting function. It thus seems that the feature responsible for the computational leap, from the computable to the "uncomputable", is the supertask; this supertask feature, it appears, is the only difference between *RM* and a "standard communication" between Turing machines.

But performing a supertask is not the only difference. Let us look more carefully at the computational structure of *RM*. One feature of *RM* is that it always reaches an end-stage. After two moments of time, the halting state of the simulated machine is displayed in the designated output cell of T_A; at this moment, we recall, T_B no longer exists. A second feature of *RM* is that it consists of two machines, T_A and

T_B, and some communication devices. Thus when talking about the configurations of *RM*, we must take into account the configurations of T_A and T_B. We cannot ignore the configurations of T_B, for example; if we do, we cannot say that *RM* performed a supertask and computed the halting function. Likewise, we have to take into account the communication between the machines. In particular, we have to take into account how and when the configuration of one machine determines the configuration of the other machine.

A third feature of *RM* is implied by the first two. It is that the end stage of *RM* cannot be described as a stage α+1, whose configuration is completely determined by the preceding stage, α. This is simply because there is no such preceding stage, α, at least when the simulated machine never halts. What would be such a preceding stage α? It cannot be the initial, "print '0'" stage, as it was followed by infinitely many stages of T_B. And it cannot be some stage of T_B, as each such stage was followed by infinitely many others. One could stipulate a "decision moment" of T_A, in which it is decided whether to replace the '0' by '1' (as in Shagrir and Pitowsky 2003); but this is just to shift the problem to the "decision moment"; now it is this "decision moment" that cannot be described as a α+1 stage.

We can conclude that *RM* cannot be described in terms of "standard communication" between Turing machines. When the communication is standard, then each stage of the communicating machines is a α+1 stage whose configuration is completely determined by that of the previous stage α (only the initial stage need not satisfy the requirement); see, for example, the very general and detailed definition of Gandy (1980), who describes the parallel computation of unbounded computing units, some even using overlapping resources. In fact, we could describe the end-stage of *RM* as some sort of a limit stage. As in an infinite-time Turing machine, the output

value ‘0’ is “calculated” as a limit of the values that this output displayed in the preceding stages; at each such stage the value is ‘0’ as long as no signal was sent after each of the infinitely many stages of T_B that preceded the end-stage of *RM*. Thus *RM* can be seen as yet another variant of an infinite-time Turing machine.

One way or another, it now seems, yet again, that the hyper-computational power of *RM* has little to do with supertasks; it has to do, rather, with the fact that *RM* has an end-stage. On the one hand, if we remove the end-stage of *RM*, as in the original AUTM, then we shall have “standard communication” between Turing machines. The *RM*-minus-the-limit-stage machine still performs a supertask: T_B still goes through infinitely many steps. Yet *RM* no longer computes the halting function: it has no end-stage in which it returns the value ‘0’. On the other hand, it seems that we can easily describe the setup of *RM* in transfinite time (or just using ordinals), perhaps in terms of infinite-time Turing machines. This newly specified machine has the same computational structure as *RM*, and it computes the halting function, yet it does not perform a supertask.

6. Why supertasks?

Supertasks are intellectually fascinating. They are discussed extensively nowadays in the context of computing machines. But why do theoreticians and practitioners find supertasks so appealing if they make no computational difference? Two conjectures are put forward here. One is that supertasks are motivated by epistemic principles. Mainly, supertasks increase our *knowledge* about the computational properties, e.g., halting states, of some machines. That there are constructions of machines that compute the halting function in transfinite time is theoretically intriguing but

practically futile. We want machines that are *useful to us*, ones that not only provide answers *in principle*, but ones that provide answers in real time. Supertasks are useful in this context in that they can provide answers beyond the Turing limit in a finite span of time. As such, supertasks increase our knowledge in ways that other theoretical constructions, such as infinite-time machines, do not (There are those who will resist the idea that supertask machines generate new pieces of knowledge. The objection is that knowledge requires the surveyability of all the steps of computation; but obviously we cannot survey the infinitely many computation steps and see that they were executed in a proper way. We address this issue elsewhere, arguing that the demand of surveyability is too strong in this context; see Shagrir and Pitowsky 2003.)

The other conjecture is that supertasks are related to a constraint on physical computation. This conjecture is motivated by the fact that supertasks are invoked in the context of *physical* machines. There is always a lengthy and lively discussion about the *physical feasibility* of supertask machines; see, e.g., Earman and Norton (1993) and Andréka, Németi and Németi (2009) for a discussion of relativistic machines. This might suggest that supertasks have to do with a constraint on *physical computation*. The constraint is that a *physical* computer must complete its task in a finite physical time. There is no such constraint on a computing system in general. We have no difficulty conceiving of the infinite-time Turing machines as computing machines even though they complete some tasks in transfinite time (Löwe, 2001, Hamkins 2002). But it might be the case that the notion of physical computation does not allow the task to be completed in infinite time; it must be completed in a finite span of time. Supertasks are useful in this context as they allow hard problems to be completed that no *physical machine* can compute without performing a supertask.

If this conjecture is valid, it has interesting implications for the notion of a physical computer. Though never defined precisely, the assumption has been that a physical computer is just a computing system *that acts in accordance with the laws of nature*. Put differently, a physical computer should satisfy two sets of constraints: (a) it should be a *computing* system, whatever that means; and (b) it should be a *physical* system, obeying the principle of Newtonian mechanics and/or of General Relativity and so forth. But the conjecture, if valid, suggests that there is another constraint to be satisfied. The additional constraint is that the machine, as a physical system, should complete the task, when the output is defined, in a finite span of time. This constraint is not imposed on computing machines in general, e.g., infinite-time machines. It is also not related to laws of nature; after all, we do let (ideal) machines run to infinity when they do not halt, e.g., the machine T_B in the relativistic case. Thus if the conjecture is valid, we invoke supertasks in order to satisfy this additional constraint on physical computation. Supertasks are invoked when our physical computer performs a task whose completion requires infinitely many computation steps.

The two conjectures are related. The first, epistemic, conjecture is about computers we can use, presumably physical ones. The second conjecture is about a constraint on physical computers. But there are also some differences. The epistemic conjecture does not imply that a machine is a physical computer only if it completes the task in finite physical time. It does not rule out physical machines that compute, say, in transfinite time. The conjecture, rather, is that we have a special interest in those physical computers that complete the task in a finite span of time. Our special interest stems from epistemic reasons, namely, that we want to know the answer. Supertasks are invoked in this context not because they make something a physical

computer, but because they make some physical computers useful for us; supertasks enable us to access the outputs of these machines.

The second conjecture makes the further step, suggesting that the completing-the-task-in-finite-time constraint is an ontological constraint. The conjecture is that a machine cannot possibly be a *physical computer* unless it completes the task in a finite span of time (when the values are defined). In particular, a physical machine computes the halting function only if it displays the values of the function in a finite physical time. According to this conjecture, supertasks allow some physical machines to be computing machines; without performing a supertask, a physical machine cannot compute the halting function.

Lastly, one might suggest that supertasks make a computational difference. The conjecture is that a computer must return the values of the function in finite time. The abstract idea of a transfinite computing is important for theoretical purposes. But we have real computing (ontologically speaking) only if we can embed this theoretical idea in some (even if idealized) physical process. The suggestion, in other words, is that we cannot detach the abstract idea of computing from a potential physical realization. The distinctions between epistemic and ontological computing and/or abstract and physical computing are not as clear-cut as presented above. The concept of supertask plays an important role in making the theoretical concepts of hyper-machines real computers, and, in this sense, increases computational power. Responding to this suggestion requires a further analysis of the concept of computing, which is not within the scope of this paper. I will just note that this suggestion does not undermine my main thesis that the leap from Turing machines to hyper-machines requires a change in computational structure. What it shows is that the "right kind" of computational structure does not suffice; you also have to embed this structure in a

supertask process. If this conjecture is correct, we can only conclude that supertasks *alone* do not increase computational power. We leave it to the reader to decide about the viability of the three conjectures.

7. Summary

We have assessed the assumption that supertasks increase computational power. We examined several instances of supertask machines. We argued that the accelerating Turing machines without a limit stage compute exactly the same functions computed by their non accelerating counterparts; in particular, they do not compute beyond the Turing limit. We then examined the accelerating Turing machines with a limit stage. These machines compute beyond their Turing limit, but not beyond the limit of their non-accelerating counterparts, e.g., infinite-time Turing machines. We extended the argument to other supertask machines, examining in more detail the case of relativistic machines. We finally attempted to locate the significance of supertasks in the epistemic and physical aspects of computing. Our arguments focused on the halting function alone, but we believe that the arguments are general and extend beyond the halting set.

Acknowledgment: Thanks to the participants of the *Workshop on Physics and Computation* at Ponta Delgada (2009) and to three anonymous referees for corrections and suggestions. This research was supported by the Israel Science Foundation, grant 725/08.

Oron Shagrir
Philosophy and Cognitive Science
The Hebrew University of Jerusalem

References:

Andréka, H., Németi, I. and Németi, P. 2009: General relativistic hypercomputing and foundation of mathematics. *Natural Computing* 8: 499-516.

Beggs, E.J. and Tucker, J. V. 2006: Embedding infinitely parallel computation in Newtonian kinematics. *Applied Mathematics and Computation* 178: 25-43.

Benacerraf, P. 1962: Tasks, super-tasks, and the modern elastics. *Journal of Philosophy* 59: 765-784.

Blake, R.M 1926: The paradox of temporal process. *Journal of Philosophy* 23:645-654.

Boolos, G.S. and Jeffrey, R.C. 1980: *Computability and Logic* (2nd edition). Cambridge University Press.

Calude, C.S. and Staiger, L. 2009: A note on accelerated Turing machines. *Centre for Discrete Mathematics and Theoretical Computer Science Research Reports* http://hdl.handle.net/2292/3857

Cohen, R.S. and Gold, A.Y. 1978: ω-computations on Turing machines. *Theoretical Computer Science* 6: 1-23.

Copeland, J.B. 1998a: Even Turing machines can compute uncomputable functions. In C. Claude, J. Casti, M. Dinneen (eds.), *Unconventional Models of Computation*. Springer-Verlag, 150-164.

Copeland, J.B. 1998b: Super Turing-machines. *Complexity* 4: 30-32.

Copeland, J.B. 2002: Accelerating Turing machines. *Minds and Machines* 12: 281-300.

Copeland, J.B. and Shagrir O. forthcoming: Do accelerating Turing machines compute the uncomputable?

Davies, B.E. 2001: Building infinite machines. *British Journal for the Philosophy of Science* 52: 671-682.

Earman, J. and Norton, J. 1993: Forever is a day: Supertasks in Pitowsky and Malament-Hogarth spacetimes. *Philosophy of Science* 60:22-42.

Etesi, G. and Németi I. 2002: Non-Turing computations via Malament-Hogarth space-times. *International Journal of Theoretical Physics* 41: 341-370.

Fearnley, L.G. 2009: On accelerated Turing machines. Honours thesis in Computer Science, University of Auckland.

Fraser, R. and Akl, S.G. 2008: Accelerating machines: A review. *International Journal of Parallel Emergent and Distributed Systems* 23: 81-104.

Gandy, R. 1980: Church's thesis and principles of mechanisms. In J. Barwise, H.J. Keisler and K. Kunen (eds.), *The Kleene Symposium*. North-Holland, 123-148.

Hamkins, J.D. 2002: Infinite time Turing machines. *Minds and Machines* 12: 521-539.

Hamkins, J.D. and Lewis, A. 2000: Infinite time Turing machines. *Journal of Symbolic Logic* 65: 567-604.

Hogarth, M. 1992: Does General Relativity allow an observer to view an eternity in a finite time? *Foundations of Physics Letters* 5: 173-181.

Hogarth, M. 1994: Non-Turing computers and non-Turing computability. *Proceedings of the Philosophy of Science Association* 1: 126-138.

Hogarth, M. 2004: Deciding arithmetic using SAD Computers. *British Journal for the Philosophy of Science* 55: 681–691.

Laraudogoitia, J.P. 2009: Supertasks (revised version). *Stanford Encyclopedia of Philosophy*, http://plato.stanford.edu/entries/spacetime-supertasks/

Lewis, H.R., Papadimitriou, C.H. 1981: *Elements of the Theory of Computation*. Prentice-Hall.

Löwe, B. 2001: Revision sequences and computers with an infinite amount of time. *Logic and Computation* 11: 25–40.

Pitowsky, I. 1990: The physical Church Thesis and physical computational complexity. *Iyyun* 39: 81-99.

Potgieter, P.H. and Rosinger, E. 2009: Output concepts for accelerated Turing machines. *Centre for Discrete Mathematics and Theoretical Computer Science Research Reports*, http://hdl.handle.net/2292/3858.

Russell, B.A.W. 1915: *Our Knowledge of the External World as a Field for Scientific Method in Philosophy*. Open Court.

Schaller, M. and Svozil, K. 2009: Zeno squeezing of cellular automata. *arXiv:0908.083*.

Shagrir, O. 2004: Super-tasks, accelerating Turing machines and uncomputability. *Theoretical Computer Science* 317: 105-114.

Shagrir, O. and Pitowsky, I. 2003: Physical hypercomputation and the Church–Turing thesis. *Minds and Machines* 13: 87-101.

Stannett, M. 2004: Hypercomputational models. In C. Teuscher (ed.), *Alan Turing: Life and Legacy of a Great Thinker*. Springer-Verlag, 135-157.

Steinhart, E. 2002: Logically possible machines. *Minds and Machines* 12: 259-280.

Steinhart, E. 2003: The physics of information. In L. Floridi (ed.), *The Blackwell Guide to the Philosophy of Computing and Information*, Blackwell Publishing, 178-185.

Stewart, I. 1991: Deciding the undecidable. *Nature* 352: 664-665.

Svozil, K 1998: The Church-Turing thesis as a guiding principle for physics. In C. Calude, J. Casti, and M. Dinneen (eds.), *Unconventional Models of Computation*, Springer-Verlag, 371-385.

Svozil, K 2009: On the brightness of the Thomson lamp. A prolegomenon to Quantum recursion theory. In C.S. Calude, J.F. Costa, N. Dershowitz, E. Freire and G. Rozenberg (eds.), *Unconventional Computation. Lecture Notes in Computer Science, Vol. 5715*. Springer, 236-246.

Turing, A.M. 1936: On computable numbers, with an application to the Entscheidungsproblem. *Proceedings of the London Mathematical Society* (2) 42: 230-265. Correction in 43 (1937) 544-546. Reprinted in Martin Davis (ed.) *The Undecidable: Basic Papers on Undecidable Propositions, Unsolvable Problems and Computable Functions*. Raven (1965), 115-154. Page numbers refer to the 1965 edition.

Welch, P.D. 2008: The extent of computation in Malament-Hogarth spacetimes. *British Journal for the Philosophy of Science* 59: 659-674.

Weyl, H. 1949: *Philosophy of Mathematics and Natural Science*. Princeton University Press.