Hypertasks Revisited

"A number of authors have discussed whether a supertask can be performed, that is, whether one can perform infinitely many tasks in a finite time. Benacerraf concluded that no cogent argument had yet shown that a supertask could not be performed, nor indeed a super-dupertask[…] Given Benacerraf's contention, a natural question arises: is it possible, in a finite time, to perform not merely countably, but uncountably many distinct tasks? Let us call such a task, consisting of uncountably many subtasks, a hypertask. We shall show that no hypertask can be performed."
-- P. Clark and S. Read, "Hypertasks" Synthese 61, 1984, pp387-390

1.

In this paper I shall be defending the possibility of performing hypertasks against the proof to the contrary presented by Clark and Read in the paper quoted. First let us examine the proof, which I have recounted here in a slightly more formal flavour for the sake of clarity.

The proof proceeds by contradiction; let us suppose that a hypertask is possible. We may label the set of subtasks that are performed H. From the definition of a hypertask it follows that

1. $|H| > \aleph_0$

Since a hypertasks must occur within a finite interval of time, T, we may partition T into sets of closed-open intervals, S, where:

a. $\forall x \in S, x \neq \varnothing$
b. $\forall x, y \in S, x \neq y \Rightarrow x \cap y = \varnothing$
c. $\bigcup S = T$

Given the function $f: H \rightarrow \wp(T)$, where f(h) for subtasks $h \in H$ is the interval in which h is performed in, we assume there exists an S* of the above form such that S* = im(f). The second premise is therefore:

2. There is a suitable partition, S*, of T such that $f: H \rightarrow S^*$ injectively.

We may also assume:

3. $\exists g: S \rightarrow Q$ where g is injective

Using a theorem from real analysis, that for any $a, b \in IR$ $a < b$, there exists a $c \in Q$ such that $a < c < b$. We may conclude that for any $x = [a, b) \in S$, there exists a rational $c \in Q \cap T$ with $c \in [a, b)$. Now we let $g(x) := c$. It follows that g is injective since S satisfies b.

4. $|Q| = \aleph_0$

This follows from a simple bijection between N and Q.
By the injection given by 2 we get,

5. $|H| \leq |S|$

By the injection given by 3 we get

6. $|S| \leq |Q|$

By 1, 4, 5, and 6 we get

7. $\aleph_0 = |Q| \geq |S| \geq |H| > \aleph_0$

Which is a contradiction.

Let us first note that the contradiction above appears to follow validly from premises (1) and (2), and that the only substantial assumption is in the construction of the function f in premise (2). Let us concentrate on this assumption. One might certainly submit to the possibility of partitioning T in such a way that each subtask of H is performed entirely in precisely one interval in the partition. However, the stronger claim that each interval in this partition contains precisely one subtask of H does not seem to follow so obviously. So, is it true that the subtasks of any supertask will embed disjointly into the time interval in which they occur? Might there not be processes we can reasonably call supertasks that disrespect this constraint? What such a possibility would amount to is the existence of infinite sequences of actions which occur in a finite interval of time T deserving to be called supertasks, in which *it is not possible to partition T in such a way that one and only one action occurs in each interval in the partition of T*.

Perhaps the constraint in (2) is just part of what it *is* to be a supertask, and I am just quibbling with definitions. Be this as it may - but permit me to abuse language a little: I shall call a supertask serial iff it respects (2), and call a supertask parallel iff not. The proof presented above then shows the impossibility of serial hypertasks. But this raises an interesting question about parallel hypertasks: should one concede that they even be considered supertasks at all, are parallel hypertasks possible?

2.

Before relaxing the constraint that seriality imposes, we must be careful that we do not allow processes which would not normally be classified as supertasks to slip past us and trivialise our claim. Indeed, once we have dispensed with the stipulation that all supertasks are serial, a more detailed analysis of what constitutes a supertask, and even a task, is now a matter of great urgency. As soon as we reject the notion that all supertasks are serial we allow processes such as simple motion to count as hypertasks thus trivialising the claim that hypertasks are possible. The reason for this is as follows: assuming space is continuous (as opposed to merely dense or discreet) we may define a hypertask H as such: every time we pass a spatial point we count this as a subtask of H. Then moving in a straight line between two positions, A and B, a finite distance apart at a constant non-zero velocity would require an uncountably infinite number of points to have been passed. Hence a trivial non-serial hypertask would have been performed (we cannot partition the time taken into intervals containing only one task each). Indeed, this appears to be the motivation which led Clark and Read, perhaps even tacitly, to embrace seriality. I would agree with Clark and Read that simple motion should not be considered a hypertask - the reason they provided was that a task should take a non-zero length of time to be performed, even if there is no lower bound on that length. This leaves us with a restriction on what may be counted as a task, namely:

α. A task must take a non-zero length of time to perform

This would dispense with the objection that motion would count as a hypertask since the act of passing a spatial point takes no stretch of time at all to perform. However, we are still confronted with another type of process which would appear to trivialise, not only hypertasks but supertasks in general. Suppose we asked, an infinite number of people, countable or larger, to simultaneously clap their hands at t=0 (we can ignore the practicalities of this for the time being). Each task – clapping a pair of hands – takes a finite but non-zero length of time, and thus an infinite number of tasks have been performed in a finite time. Such set-ups are clearly uninteresting as far as discussions about supertasks are concerned, and it would be desirable to exclude them by adding a further restriction:

> β. A supertask may not have infinitely many tasks being performed simultaneously at any given time during the performance of the supertask.

It is worth remarking here that this does not place any constraints on the status of the system at the first instant of time (if there is one) after the supertask has been performed. (I am not concerning myself with super-duper or hyper-dyper tasks). Indeed the status of the world after a supertask has been performed is not logically determined by the preceding states as Benacerraf has so convincingly argued[1]. This restriction also excludes processes such as the following:

"An interesting objection describes Zeno's runner as completing at every moment in the second half of his run the task of crossing half the stadium floor, and thus performing a hypertask by the completion at the end of his run of an uncountable succession of acts." -- P. Clark and S. Read, "Hypertasks" Synthese 61, 1984, p390, footnote 1.

Such a description of a supertask violates β. Clark and Read, however, reject it on other grounds:

"But this objection again falls to the reply that it is a mere analysis of a single task, for the individual tasks purportedly making up the hypertask cannot be made separate and distinct"

This lends interesting insight as to why a tacit commitment to the second premise was made, where I believe what is meant by "separate and distinct" here is that the tasks are performed serially. That supertasks must be serial is a much stronger requirement than the requirement that they must satisfy α and β. Indeed, neither of the two trivial "hypertasks" given *are* serial, and are thus conveniently excluded from consideration. However I feel the reasons we do not consider them to be hypertasks are given by α and β respectively.

3.

So having identified the offending premise, the onus is upon me to provide an instance of a hypertask. The counter-example I have in mind deserves, I believe, to be called a parallel hypertask, if not with respect to the number of tasks it has performed, then in lieu of its computational power. That is, if you should deny this is a hyper*task*, I claim it is at least capable of a hyper-computation (I shall come back to this in section four). As far as I can see this example should be perfectly acceptable to someone who accepts the possibility of run-of-the-mill countable supertasks. Furthermore it has been argued that similar supertasks are even physically possible[2]. The example I have in mind is based on parallel processing and can be described as follows. It is an infinity machine (a machine

[1] Cf. Benacerraf, P., 1962, 'Tasks, Super-Tasks, and Modern Eleatics', *Journal of Philosophy*, **LIX**, pp. 765-784; reprinted in Salmon [1970]

[2] Cf. Davies, E.B., 2001, "Building Infinite Machines", *British Journal for the Philosophy of Science*, **52** , pp. 671-682

capable of performing an infinite number of tasks in a finite time). The process starts off at time t = 0 with what I shall call the mother machine which has no memory and finite speed. At time t = ½ my machine creates two exact replicas of herself except in total they have half her size (so they are a quarter her size each), have twice her speed and have one extra bit of memory. The offspring, which have imaginatively been dubbed 1 and 0, repeat the same process at t = 1 - ¼, storing their names in their one bit of memory. In general the parent machine names its children 1 and 0, and appends to the end its own name as the child's surname. After time $t = 1 – (½)^n$ the number of machines is $2^{n+1} – 1$, the last machines to be created will be of speed 2^n times the original mothers speed, and will each have a name that is n bits long. So after $t = 1 – (½)^3$ the process will look like figure 1.

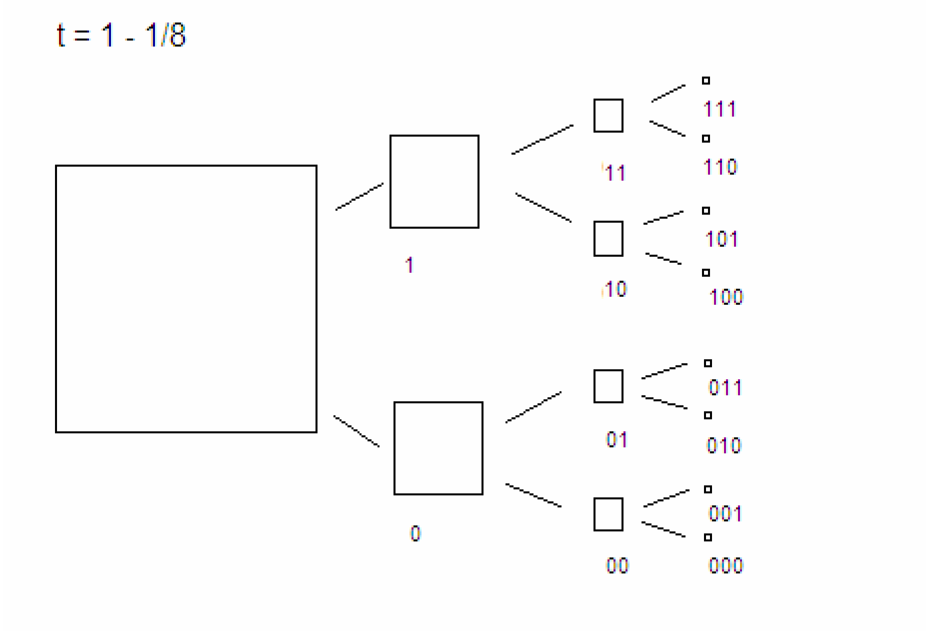

Fig 1.

Now it can be seen that each infinite branch - where a branch is a chain of adjacent ancestors – constitutes a sequence of 1's and 0's, i.e. a function $f: N \rightarrow \{0, 1\}$, namely for any $n \in N$, let f(n) be the christian name of the nth machine in the branch. Each such function is distinct since there is at least one value where they differ, namely the number n where after the nth machine the two corresponding branches split off from each other, and they exhaust the all the sequences in 0 and 1. Let a task be any finite branch (a branch being totally ordered under a temporal relation) or the union of a set of finite branches totally ordered by the initial parthood relation (the relation that holds when one branch is entirely an initial part of the other)[3]. So H contains all the distinct infinite branches, each a countable serial supertask. So |H| is at least the number of functions from N to {0, 1}, thus $|H| \geq |\{0, 1\}^N| = |\{0, 1\}|^{|N|} = 2^{\aleph_0} > \aleph_0$. We can see this result intuitively, since a sequence of binary numbers will be generated by considering the full

[3] We might have let our subtasks be just the infinite branches, but then it might be complained that at no point before t=1 had a subtask been completed, even though an infinite number had at t=1 and afterwards.

names of the sequence of machines along some branch. Each of these sequences will tend toward a distinct real number without a binary place (the equivalent of a decimal place). This method of generating real numbers from each branch will be useful later.

Firstly we must note that our hypertask obeys both α and β. Each subtask, takes a non-zero but finite amount of time to perform its task. Also, at any point during the hypertask only a finite number of subtasks are being performed in parallel. My hypertask is not a hyper-dyper-task - that is, I have not specified the end-state of my machine – and therefore we needn't be left with the physical embarrassment of an uncountable infinity of tiny machines after the minute is up. Furthermore it can be seen exactly why this violates seriality, because at time $t = ½$ there are two tasks being performed simultaneously and at $t = ½ + ¼$ there are 4 tasks being performed simultaneously and so forth. It is therefore impossible to create an interval of time which has only one task being performed in it at a time.

4.

With an ontology of temporal parts and unrestricted composition it is very easy to give counterexamples to Clark and Reads claim. Not only are there an uncountable number of 3D slices of my body's space-time worm, but also an uncountable number of overlapping 4D sausages, as seen by Zeno's alternative runner, who completes the task of running half the distance of the stadium floor an uncountable number of times, not to mention a load more gerry-mandered objects (more than $2^{\aleph_0}$ even). This might cause one to complain about my hypertask on similar grounds, since any two subtasks of my hypertask will share a common part. They might claim that α and β are not strong enough constraints and that a further stipulation is in order:

γ. The subtasks of a supertask must be mereologically disjoint.

This, together with α and β is probably enough to show that hypertasks are impossible, if not establish seriality. We can see this kind of idea in Clark and Read:

"Thus the performance of a hypertask requires an uncountable set of **separable and distinct** *tasks to be completed in a finite interval of time[…]"* P. Clark and S. Read, "Hypertasks" Synthese 61, 1984, pp387-390 (my emphasis)

I consider this a fair objection, and I have already said I do not wish to bicker over definitions. Very well. However a far more interesting issue here is the computational power of this machine.

Here I outline an interesting application of a hypertask of the form described above, for solving, by brute force, the Riemann Hypothesis. I shall miss out some of the more technical details for the sake of clarity, but at points where I have skipped details it should be seen how to make it rigorous. There is probably a simpler way of showing how this is machine is constructed, not all the branches are necessary, however it is easier with them. First we must modify the supertask outlined above, so that it enumerates the complex numbers rather than the reals[4]. Modify the machine shown in figure 1

[4] We can actually enumerate the complex numbers on the machine in figure 1, taking the odd place bits to for the real component and the even place bits for the imaginary, so the 1st, 3rd, 5th… bits represent the real part and the 2nd, 4th… represent the imaginary part, but this only complicates things.

appropriately so that at t = ½ it splits into four separate machines instead of two, and call them one 1+i1, 1+i0, 0+i1 and 0+i0, extend the process appropriately and it should be obvious we will get converging sequences for binary expansions for the whole of the complex plane. We have omitted the position of the binary place to keep the machine simple. Now let us restrict our attention to one particular branch, which corresponds to a complex number s, call this the s-branch. As noted already, an s-branch constitutes a standard countable serial supertask. To prove the Riemann Hypothesis it will be enough to show that such a supertask can evaluate the Riemann zeta function at s. This is because if for each complex number s the s-branch has evaluated $\zeta(s)$, then we have checked every complex number and may say whether or not it is a root of ζ, in particular we may say whether all non-trivial roots lie on the line y = ½ + ix. We may modify our machine further so that at $t = 1 - (½)^n$ as well as spawning four more replicas my machine evaluates the first n terms in the Taylor expansion of ζ at the complex point represented by the 2n-bit long complex number of that machine. Now as we get further down the Taylor expansion of ζ the sum of the first n terms might become harder to calculate. We can compensate for this: rather than having the n^{th} machine have 2^n times the original speed we can let it have the complexity of the Taylor expansion algorithm, O(n), times the original speed by having each machine multiply the speed of its children by the correct factor as opposed to 2, noting that for all n, $O(n) < \infty$. (Note: if we where being rigorous here we would also need to compensate for the calculation of O(n) itself).

Now communicating these values back to the mother machine in a way that she can cope with is slightly more complicated. Each machine must spawn an extra linear branch (a countable serial supertask) called the evaluation branch. The purpose of these machines is to compare increasingly large numbers to see which is bigger, each machine is capable of comparing successively larger numbers at greater speeds. The non-evaluation machines roughly follow this algorithm: each machine calculates their number and sends their findings up one level, creates two babies and goes into evaluation mode. Each machine in evaluation mode takes the two results being sent upwards and sends them down the evaluation branch (this is how the need for infinite memory is evaded) and sends the result of the evaluation up a level. Meanwhile, the machine at the end of the evaluation branch compares the two values and sends the larger one back up to the machine that spawned the evaluation branch.

The mother-machine is yet more complicated. It has an extra branch, the winners-branch. When the final winning number, s, comes out of the mother-machines evaluation branch it is sent down the winners branch, where it is compared with a number, epsilon, given to the final machine in the branch by its immediate parent (arbitrarily set to 1 at t=0). If epsilon wins the last machine creates a baby and passes epsilon onto it, if s wins then s/2 is passed onto the baby machine. Whenever epsilon loses it sends a message up to the mother-machine who flips a bit. Note that, the status of the sequence a_n given by the bit at t = 0, ½, ¾ … determines whether or not we have a counterexample to Riemann's hypothesis. If it converges then ζ has reached a non-zero minimum, and if not it has reached 0.

Finally we outline how to show whether a_n converges. We have another branch consisting of machines m_1, m_2, m_3, m_4 etc… where m_n takes term a_n and spawns a linear branch to check whether $a_n = a_{n+1}, a_{n+2}, a_{n+3}…$ If at any point it doesn't, the branch reports back. If after a certain amount of time the branch has not reported back then m_n reports back to the mother, informing her the sequence converges.

This completes the hyper-computation, and with any luck, will confirm the Riemann Hypothesis.