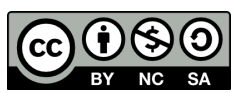
Hypercomputation—A Rant

Klaus Sutner

Carnegie Mellon University

Fall 2024



Tired of Computation?

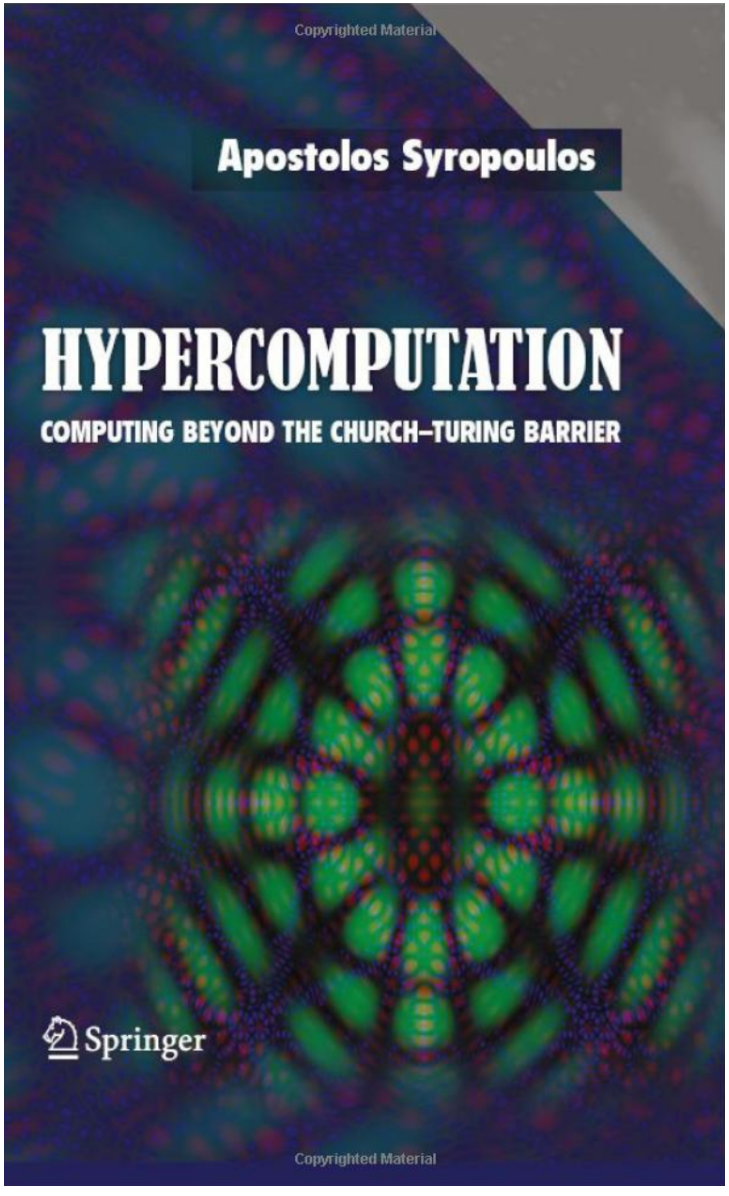
Tired of computability, complexity, algorithms, average case analysis, efficiency, non-computability, intractability?

All the endless of effort of thinking about hard problems?

No problem. There are people working in the brand-new and exciting field of

HYPERCOMPUTATION

Just wait a little bit longer, and your desktop machine will be replaced by a hypercomputer and will solve unsolvable problems on a routine basis. Merely intractable problems are handled instantaneously.


Copyrighted Material
Apostolos Syropoulos
HYPERCOMPUTATION
COMPUTING BEYOND THE CHURCH–TURING BARRIER
Springer
Copyrighted Material

At present, the theory of computation falls into two major parts:

Classical Computability Primitive recursive functions, Turing/register machines, λ-calculus, decidability, semidecidability, arithmetical hierarchy, degrees of unsolvability, ...

Complexity Theory Time and space classes, deterministic and nondeterministic classes, circuits, $\mathbb{P}$ versus $\mathbb{NP}$, randomness, probabilistic classes, quantum computation, ...

Could this be all? Nah ...

Historical Hypercomputation

This area is known as Generalized Recursion Theory (GRT) (as opposed to Classical Recursion Theory (CRT)) and has been studied extensively for almost a century, there are sophisticated methods and lots of interesting results.

Hysterical Hypercomputation

A more recent idea, unencumbered by any sort of results. It relates to actual computability theory in about the way astrology relates to astronomy.

My favorite quote from Stefan Banach:

> A mathematician is a person who can find analogies between theorems; a better mathematician is one who can see analogies between proofs and the best mathematician can notice analogies between theories. One can imagine that the ultimate mathematician is one who can see analogies between analogies.

Does this apply to computability?

Is there any reasonable way to generalize computability?

Is there any mileage one can get out of such generalizations?

A good number of theories of computation on structures other than the natural numbers have been developed: computation on words, ordinals, sets, algebraic structures, higher types and so on.

In fact, there is even an axiomatization of computation that tries to pin down the most basic principles of computation in a clear, logical form:

> J. E. Fenstad
> General Recursion Theory: An Axiomatic Approach
> Springer 1979

Unfortunately, the axiomatization by Fenstad feels a bit awkward and overly technical (compared to, say, Hilbert's axiomatization of geometry or Zermelo-Fraenkel's axiomatization of set theory), but overall it captures the fundamental ideas behind computation fairly well.

All the experts tell you to

- keep text on slides to a minimum
- under no circumstances read your own slides
- use simple, stripped down graphics
- do **not** use multiple fonts, loud backgrounds, garish colors, useless pictures, cartoons, watermarks, anything distracting . . .

In general, I try to adhere to these principles[†], but there will be too much text on the following slides, and I will read some of it aloud. Sorry.

[†] Except for item 1, I can't stand slides that are utterly meaningless without a video.

There are some areas of intellectual discourse where the experts can engage in furious debates about the basic usefulness of certain methods, even beyond the question of correctness. Psychology, philosophy, literature, history, economics and so forth come to mind.

However, in the hard sciences, these debates tend to be rare and focus on particular technical issues:

- is string theory falsifiable,
- is AI a threat to civilization,
- is automatic theorem proving essential for math.

Amazingly, we now have an example of an entirely meaningless theory in computability theory, or at least some area close thereto.


OR

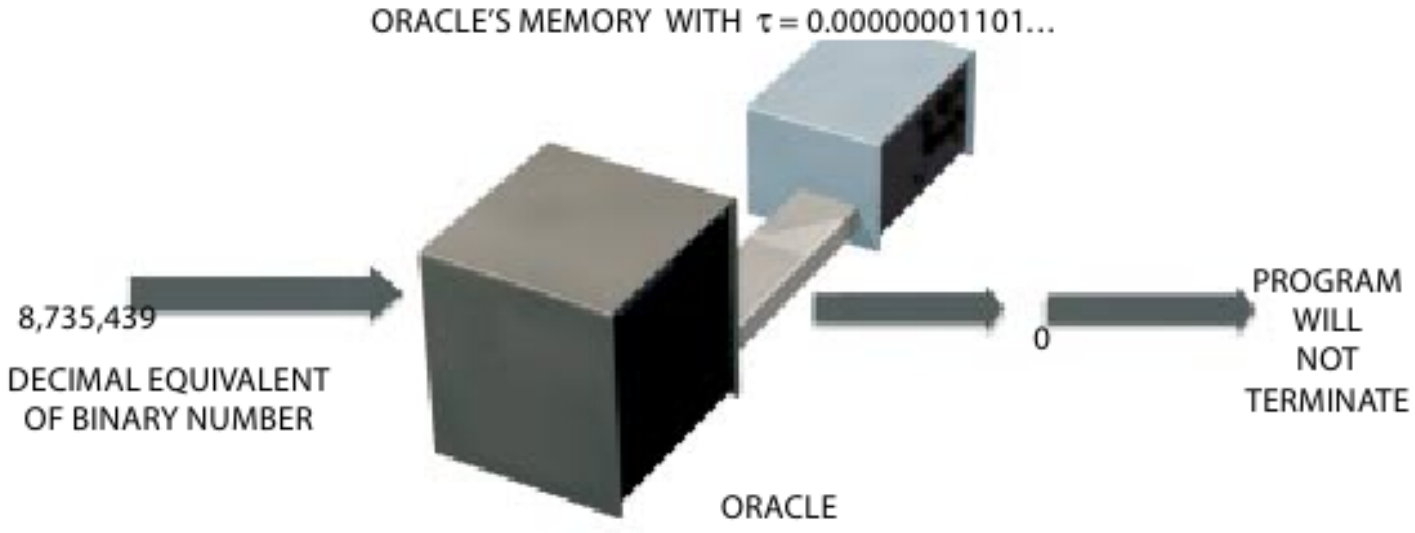
The term hypercomputation gained notoriety after a 1999 paper by Copeland and Proudfoot in the Scientific American titled

> Alan Turing's Forgotten Ideas in Computer Science

The article makes the most astounding claim that Turing "anticipated hypercomputation" in his technical work.

A clever PR trick, of course. Hiding behind one of the greats is usually a good idea—in particular if the person is dead and cannot complain.

To support this conclusion, the authors misinterpret Turing's concept of an oracle machine in patently absurd ways. To help the baffled reader, the authors provide a lovely picture of an oracle machine:



The idea is that the big, ominous, gray box (the oracle) has access to a copy of the Halting set, an infinite bit-sequence τ that lives in the smaller, more appealing blue box.

The authors comment:

> Obviously, without τ the oracle would be useless, and finding some physical variable in nature that takes this exact value might very well be impossible. *So the search is on for some practicable way of implementing an oracle*. If such a means were found, the impact on the field of computer science could be enormous.

Italics mine.

No doubt, the impact of finding τ under a rock somewhere in New Zealand could be enormous. For example, we could potentially solve Diophantine equations and get answers to all kinds of open problems like the Riemann hypothesis.

To first-order approximation, there are two basic possibilities for the resource requirements of the CPM:

- It is reasonably fast, say, low degree polynomial time.
- It is not, the running time is some rapidly growing horror.

If the CMP is slow, it is just as utterly useless with τ as without it.
One suspects this is not what the authors had in mind.

Working Assumption: The CPM is fast.

The reason Halting is quite so important is that a great many problems can be reduced to it quite easily. E.g., let $n = pq$ be the product of two large primes, so RSA depends on factoring n being hard.

With CPM, this is a piece of cake. For $2 \leq a \leq b \leq n$, construct a Turing machine $M_{a,b}$ that halts if there is some factor of n in the interval $[a, b]$. Then do a binary search.

Similar tricks would destroy just about all the cryptographic schemes relevant today (I suppose quantum-generated one-time pads would still work). So, the world financial system would implode immediately, followed by the whole world economy, and soon after we'd be back in the stone age.

The next vexing issue is that oracle τ depends on storing an infinite amount of information in a finite physical system—remember the little blue box. There are good reasons to believe that this is quite patently impossible.

And even if we could somehow store an infinite amount of information in the blue box, how could we possibly retrieve it with sufficient precision—perhaps the authors are unaware of Heisenberg's uncertainty principle?

We would have to fire photons of arbitrarily high energy at our box to get at distant bits, which would destroy the device immediately and the whole CPM would turn into a black hole.

It bothers me that, according to the laws as we understand them today, it takes a computing machine an infinite number of logical operations to figure out what goes on in no matter how tiny a region of space, and no matter how tiny a region of time. How can all that be going on in that tiny space? Why should it take an infinite amount of logic to figure out what a tiny piece of space-time is going to do?

So I have often made the hypothesis that ultimately physics will not require a mathematical statement, that in the end the machinery will be revealed and the laws will turn out to be simple, like the checker board with all its apparent complexities.

R. Feynman, 1965

A Misunderstanding?

Maybe Copeland-Proudfoot did not really mean to have an infinite oracle, perhaps they were only hoping to get the first, say, trillion bits.

A nice initial slice of τ could still have huge impact on math. Depending on coding details we might be able to resolve open problems such as the Riemann hypothesis or the Goldbach conjecture, the consistency of Dedekind-Peano arithmetic, and so on.

Implementing a bit-string of length 10^{12} would be entirely trivial, if only someone could provide the actual bits. Alas, that's exactly where things fall apart; any finite oracle exists abstractly, but we don't know how to construct it.

Suppose Zeus wakes up tomorrow and finds himself in a particularly spectacular mood. He decides to gift mankind with the first trillion bits of τ. He asks Athena to figure them out and sends Hermes down from Mount Olympus with a USB stick.

Great, right?

Not really. How do we know that the bits are correct? There is no computable way to test whether the bits have anything to do with τ—we would get answers but there would be no reason to trust any of them.

Many objections could be raised to this proposal. The most relevant for us is that *abstract mathematical entities are not the right kind of entity to implement a computation*. Time and change are essential to implementing a computation: computation is a process that unfolds through time, during which the hardware undergoes a series of changes (flip-flops flip, neurons fire and go quiet, plastic counters appear and disappear on a Go board, and so on).

Abstract mathematical objects exist timelessly and unchangingly. What plays the role of time and change for this hardware? *How could these Platonic objects change over time to implement distinct computational steps?* And how could one step "give rise" to the next if there is no time or change? Even granted abstract mathematical objects exist, they do not seem the right sort of things to implement a computation.

WTF?

The next quote is taken from a 2006 article titled "The many forms of hypercomputation" by Toby Ord, an Oxford educated thinker and one of Copeland's acolytes.

In the interest of fairness, Ord is really a moral philosopher and quite impressive as such, see Giving What We Can.

Let us suppose, however, that hypercomputation does turn out to be physically impossible–what then? Would this make the study of hypercomputation irrelevant? No. Just as non-Euclidean geometry would have mathematical relevance even if physical space was Euclidean, so too for hypercomputation.

Perhaps, we will find certain theorems regarding the special case of classical computation easier to prove as corollaries to more general results in hypercomputation. Perhaps our comprehension of the more general computation will show us patterns that will guide us in conjectures about the classical case.

All good, but this is one of the central reasons why recursion theory was generalized in the first place: to gain additional insight into the nature of ordinary computation. Sometimes one really can miss the forest for the trees.

So what exactly are the new results or insights obtained specifically from the Copeland-Proudfoot approach? As far as I can tell, none.

Also note the evolution on the issue of implementability, apparently the oracles have so far escaped capture and are still at large. Hence the need to salvage hypercomputation in the absence of physical oracles.

Thus the claims that such problems are "undecidable" or "unsolvable" are misleading. As far as we know, in 100 years time these problems might be routinely solved using hypermachines. Mathematicians may type arbitrary Diophantine equations into their computers and have them solved. Programmers may have the termination properties of their programs checked by some special software.

We cannot rule out such possibilities with mathematical reasoning alone. Indeed, even the truth or otherwise of the Church-Turing Thesis has no bearing on these possibilities. *The solvability of such problems is a matter for physics and not mathematics.*

Seems like implementability is critical again, all of a sudden. Instead of gaining alternative views on general issue of computability, we are back to having to actually build and run our hypermachines.

The idea that undecidability is a matter of physics, not math, is just utterly insane. In Ord's defense, there are some notable physicists that have made similar wrongheaded claims (Landauer, Deutsch). They all fail to comprehend the difference between classical recursion theory and physical realizability.

Alas, that is a topic for another rant.

This is not even wrong.

Just to demonstrate that it can be perfectly interesting and productive to generalize ordinary computation, here is one example that is intuitively quite appealing:

Infinite Time Turing Machine (ITTMs)

Usually we want a Turing machine to stop after finitely many steps[†] so we can read off the result and go on to do other things.

However, it also makes sense to have a TM run forever.

For example, we could use the standard dovetailing approach to construct a TM that runs forever and writes all e such that $\{e\}$ halts on empty tape on a special output tape.

This is perfectly fine intuitively, TMs can enumerate certain infinitely many steps. Alas, to get real mileage out of this idea, one needs to work a bit harder to make the notion of "run forever" more precise.

[†] Ironically, in Turing's 1936 paper they were not supposed to stop.

Technically, we need two main ingredients to formalize our idea:

- The use of transfinite ordinals rather than just natural numbers to count steps in a computation.
- A mechanism to preserve information when a computation reaches a limit stage, without breaking the constraint that a machine has to be a strictly finitary object (at least the control mechanism).

To be clear: we wind up with a model of computation that is eminently unrealizable, we cannot build such a device within the framework of physics. Still, it fits in nicely with other ideas from generalized recursion theory, and the ultimate justification is that a number of interesting results can be obtained this way.

It is easy to add another step to any computation, the real problem is to deal with limit stages when infinitely many steps have just been performed. Here is the basic idea.

We can associate the steps of an ordinary Turing machine with the natural numbers:

$$0, 1, 2, \ldots, n, n{+}1, \ldots \mid$$

The notation $\ldots \mid$ is meant to indicate infinitely many steps, as opposed to the first $\ldots$, which stands for finitely many steps[†]. We write ω for all these infinitely many steps.

The idea is that a machine, after running through all the finite steps n, arrives at the (first) infinite level ω. To deal with endless boredom, the machine goes into a trance along the way, and wakes up at level ω.

[†] The ellipsis is one of the most consistently abused symbols in all of math.

So suppose our machine has reached level ω. Nothing can stop us from taking another step, leading to level $\omega + 1$. And then to $\omega + 2$, . . . , $\omega + n$, and so on. If we keep going, we finally wind up at

$$0, 1, 2, \ldots, n, \ldots \mid \omega, \omega + 1, \omega + 2, \ldots, \omega + n, \ldots \mid$$

which we express as $\omega + \omega = \omega \cdot 2$. So these are two infinite blocks of activity, one after the other.

You guessed it, we can also get $\omega + \omega + \omega = \omega \cdot 3$. In fact, we can get

$$\omega, \omega \cdot 2, \omega \cdot 3, \ldots, \omega \cdot n, \ldots \mid$$

We denote this level by $\omega \cdot \omega = \omega^2$: ω many blocks of size ω each.

In a similar way we can get to higher powers ω^k and ultimately to ω^ω.

Moving right along, we get ω^{ω^ω} and so on.

Nothing can stop us now, we get to level

$$\varepsilon_0 = \omega^{\omega^{\omega^{\cdot^{\cdot^{\cdot}}}}}$$

So this is a stack of ω many ωs all exponentiated somehow.

We'll stop here to avoid injury to malleable young minds. But rest assured, one can keep on going on, and on, and on . . . |

Aside: Induction up to ε_0 is needed to prove the consistency of Dedekind-Peano arithmetic, your favorite system from 15-151.

If transfinite ordinals sound alluring, but perhaps a bit hallucinatory, no worries.

Von Neumann showed how ordinals can be constructed in a perfectly rigorous and formal way in, say, Zermelo-Fraenkel set theory.

Moreover, one can use a transfinite version of recursion to define all the required arithmetic operations (addition, multiplication, exponentiation, ...) on ordinals. They really form a nice extension of the natural numbers.

The next key questions is: how can we make sense of the computation of a Turing machine at stage ω? Or any other limit stage for that matter?

There is a clever model developed by J. Kidder, J. Hamkins and others. We use a Turing machine with a one-way infinite tape that is subdivided into three tracks:

input	u_0	u_1	$\ldots$	u_n	$\ldots$
scratch	x_0	x_1	$\ldots$	x_n	$\ldots$
output	v_0	v_1	$\ldots$	v_n	$\ldots$

We can safely assume that each tape cell (a single column on the tape) contains 3 bits (u_i, x_i, v_i). We have a finite state control and a read/write head, as usual.

Now suppose the Turing machine has already performed all steps $n < \omega$. We define the configuration at time ω as follows:

- The machine is in a special state $q_{\lim}$.
- The head is in position 0.
- The content of each subcell such as x_i is the limsup of its contents at times $n < \omega$.

The definition for an arbitrary limit level λ is exactly the same. Thus, the entry in a subcell is 1 at time λ iff

$$\forall\, \beta < \lambda\, \exists\, \alpha \left(\beta < \alpha < \lambda \ \wedge \ \text{symbol at time } \alpha \text{ is } 1 \right)$$

Think of a light blinking infinitely (actually, cofinally) often before time λ.

Note that the special limit state $q_{\lim}$ is the same for each limit λ; there is no special q_ω, $q_{\omega+\omega}$, $q_{\omega\cdot\omega}$ and so on. So the state set of a ITTM is still finite.

The only way to preserve information at a limit stage is our limsup mechanism; state and head position are always exactly the same when we wake up on the other side.

Essentially, we have a a way to check whether a particular condition was met over and over again, arbitrarily close to the the limit stage.

Clearly, there is no analogue to this in an ordinary Turing machine, it dies at level ω and, sadly, never wakes up again.

Example: Prime Twins

We can use an ITTM to check whether there are infinitely many prime twins.

foreach $n \in \mathbb{N}$ **do**
    if n and $n+2$ are prime
    then flash x_0 // turn on and then off again right away

if $x_0 = 1$ // we are at a limit stage
then Yes
else No

This machines solves the prime twin conjecture in $\omega + 1$ steps.

Example: Halting

Consider the Halting problem for ordinary Turing machines: given an index e we want to know if $\{e\}$ halts on the empty tape.

This can be handled by an ITTM $\mathcal{M}$ where e is written on the input track.

- $\mathcal{M}$ simulates the computation of $\{e\}$ on empty tape.
- If $\{e\}()$ halts after finitely many steps, $\mathcal{M}$ also halts and accepts.
- Otherwise $\mathcal{M}$ wakes up in state $q_{\lim}$.
  It takes one more step, halts and rejects at time $\omega + 1$.

Note that all we need here is to reach state $q_{\lim}$, the limsup mechanism is not required.

Example: All of Halting

We can do better than this: we can dovetail all computations $\{e\}()$, $e \in \mathbb{N}$, and write a 1 in position e of the output track whenever the corresponding computation converges.

At time ω, the (characteristic function of the) Halting set will appear on the output track.

More generally, we can use ITTMs to compute functions

$$f : \mathbf{2}^{\mathbb{N}} \longrightarrow \mathbf{2}^{\mathbb{N}}$$

essentially functions on the reals.

Halting is near the bottom level of the arithmetical hierarchy, but we can also handle higher levels via ITTMs.

For example, recall INF, the collection of all Turing machines that halt on infinitely many inputs:

$$\text{INF} = \{\, e \in \mathbb{N} \mid \{e\} \text{ converges on infinitely many inputs} \,\}$$

Intuitively, this is clearly harder than plain Halting.

In fact, we have seen INF is Π_2-complete in the arithmetical hierarchy. Pretty hopelessly hopeless ...

Lemma

We can decide membership in INF *by an ITTM.*

Just to be clear: we are making no claim that INF is decidable, it's definitely not. But it is ITTM-decidable, we are dealing with much stronger model of computation.

Unlike other models in generalized recursion theory, ITTMs are a rather neat and require far fewer technicalities than more traditional approaches.

As before, e is written on the input track.

Then ITTM $\mathcal{M}$ runs the following program:

foreach $n \in \mathbb{N}$ **do**
    if $\{e\}(n)\downarrow$
    then flash x_0 // turn on and then off again right away

if $x_0 = 1$ // we are at a limit stage
then accept
else reject

So $\mathcal{M}$ flashes a light whenever it finds a convergence. Infinitely many flashes means e is in INF.

If $\mathcal{M}$ finds that $\{e\}$ converges on n, it turns bit x_0 on, and then off again at the next step. Of course, $\mathcal{M}$ will not use subcell x_0 for the simulation, just for messaging.

If the machine $\{e\}$ diverges on n, we just spend ω many steps finding out, without ever flashing x_0.

At the limit stage corresponding to completion of the main loop, subcell x_0 will hold a 1 iff there were infinitely many good arguments n.

The check for each n may require up to ω steps, so the total running time is between ω and ω^2.

Here is a nice example of a theorem for ITTMs that exhibits a new type of behavior. This is the kind of result that makes GRT worthwhile.

An ordinary Turing machine can halt, enter a loop or diverge (run through an ω-sequence of non-repeating configurations). By contrast, an ITTM either halts or enters a loop of sorts: even if it "diverges" for a while, it will ultimately end up in a limit cycle. How far do we have to go before the computation ends in this sense?

Theorem

Every computation of a ITTM either halts or enters a loop after countably many steps.

So we do not have to run the machine $\aleph_1$ or $\aleph_{17}$ many steps, some $\alpha < \aleph_1$ will do. Small consolation, but at least we are not totally out to lunch.

To summarize: Classical recursion theory (CRT) can be generalized. And it has been, for nearly a century. There are many models $\mathcal{M}$ that are both mathematically sound and define $\mathcal{M}$-computations that are more or less powerful than ordinary Turing machines[†].

Generalized recursion theory (GRT) is interesting at the very least since it sheds light on the fundamental ideas in ordinary computability. And there are connections to other areas of mathematics (e.g. set theory).

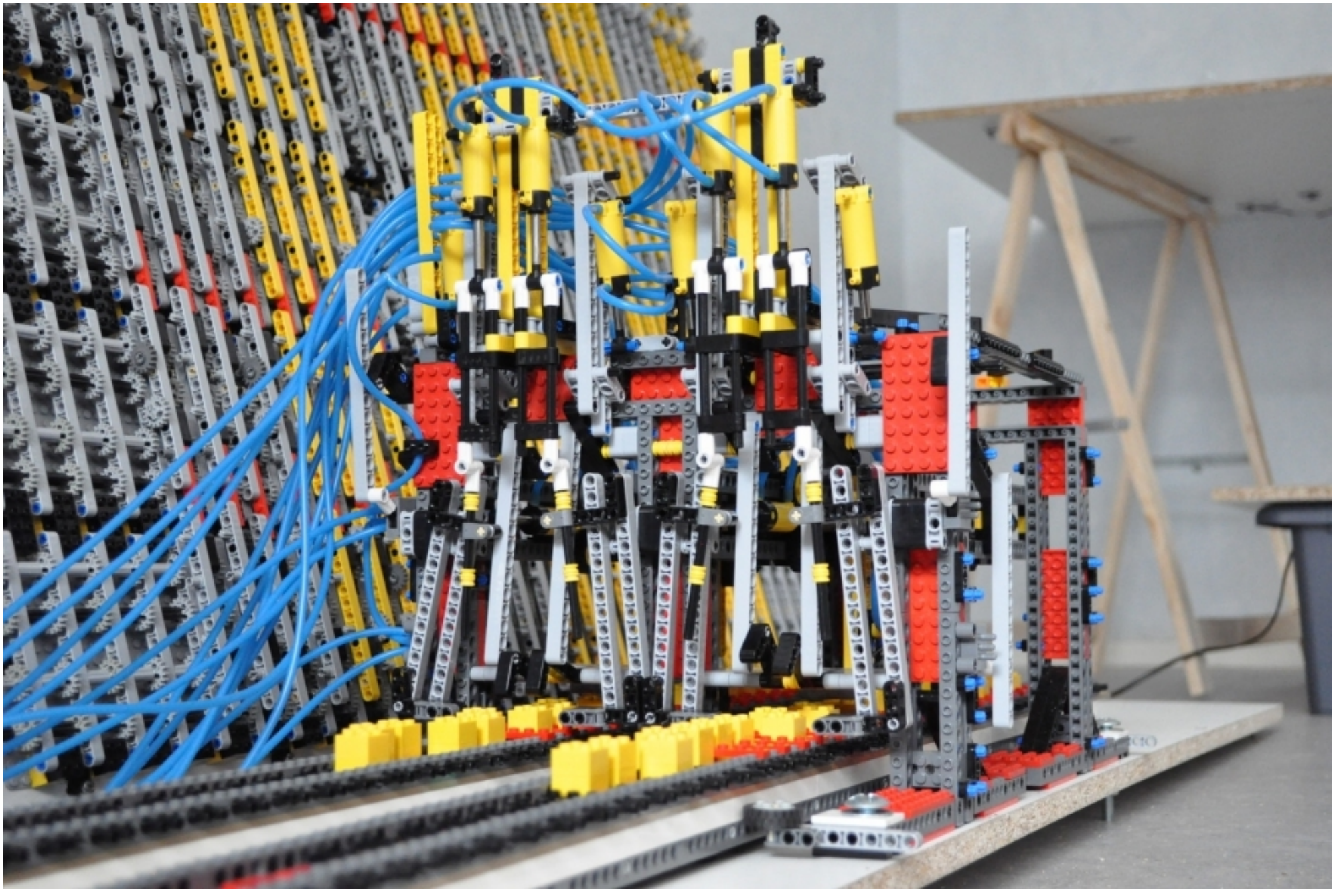
To breath even a semblance of life into hypercomputation on needs to cling to physics and implementability for dear life: otherwise one winds up with a particularly insipid and boring form of GRT.

[†]The less powerful ones are hugely important in computer science, both at the applied end and in abstract theory

Once physics is taken into consideration one is confronted with a beautiful, albeit brutally hard, question:

> Can one physically implement computations that go beyond CRT?
> In this here universe that we currently occupy?

One of the great attractions of Turing machines (as opposed to the λ-calculus or Herbrand-Gödel equations is that they are obviously physically realizable. In fact, people have built LEGO Turing machines.



One can clearly build physical devices that simulate small Turing machines performing small computations.

Alas, there are many computable functions such as Ackermann's monster that can be computed by a TM[†], but these computations cannot be realized in physical reality. Not in this particular universe, at any rate.

Hence, in any exposition that is carefully phrased, one always finds a hedge along the lines of:

> In principle, Turing machines are physically realizable. However, resource limitations . . .

[†] The Turing machine for Ackermann is actually not particularly large; this is a characterbuilding exercise.

So how would we **prove** or refute the claim that physics can implement some hypercomputation or other?

The CPM is utterly useless in this regard, it relies on the realizability of some magic bit string $\tau \in \mathbf{2}^{\omega}$—without even the slightest attempt to justify the assertion that such a string can somehow be implemented.

The only weight-bearing way is to

- Axiomatize physics in some formal system Γ, in some suitable logic.
- Show (preferably using a theorem prover) that

$$\Gamma \vdash \exists M \left(\text{device } M \text{ solves Halting}\right)$$

Hilbert alluded to the axiomatization question already in his famous 1900 address.

> The investigations on the foundations of geometry suggest the problem: To treat in the same manner, by means of axioms, those physical sciences in which already today mathematics plays an important part; in the first rank are the theory of probabilities and mechanics.

Unfortunately, Hilbert's problem $\#6$ (interpreted as concerning all of physics), is still unanswered today: no one knows how to axiomatize physics in its entirety. We would need a Theory of Everything (ToE) and that seems brutally difficult.

Suppose someone proposes a ToE Γ that seems to work and passes all the experimental tests we can throw at it. If no one can come up with a bad experiment for, say, 100 years the theory is good.

As Karl Popper happily pointed out, this method may be very appealing, but it it is really quite weak from a logical perspective. There is nothing like a inductivist correctness proof in physics, there are only depressing falsifications.

Ultimately we are simply not dealing with a problem in pure computability theory, and there is no reason to believe that the methods that work perfectly well there can nicely be carried over to include physics.

Just to be clear, exploring the computational aspects of existing physical theories, even partial or inaccurate ones, is not all useless, far from it.

In fact, it is an excellent exercise to fix some particular theory Γ of physics (not a ToE) and try to show that in Γ it is possible to "construct a device" that solves the Halting problem—or prove that it cannot be done.

For example, assume Newtonian physics: gravitating point masses, no relativity theory, no quantum theory. Then build a hypercomputer that solves Halting.

So What's The Difference?

CPMs solve the Halting problem, as do ITTMs. Why should one care about one model but not the other?

Because ITTMs produce an interesting theory of computation that has close connections to other areas of generalized recursion theory, along the lines discussed above. The proof techniques are interesting and quite complicated.

CPMs, on the other hand, are utterly useless, just a shallow public relations stunt that is of no importance to mathematics, computer science or physics.

Hodges has written the definitive biography of Turing (incidentally, very well worth reading). He found it necessary to comment on the Copeland/Proudfoot article, an unusual step in the genteel world of math.

> I was appalled that this hare-brained idea should be associated with Alan Turing as his 'lost brainstorm.' Scientific American said that this 'hyper-computation' is a 'hot idea' which Alan Turing had 'anticipated in detail.' I suspect many people with a physical or engineering background took it, on reading this nonsense, that Turing had come up with this ridiculous idea because he was an impractical logician without understanding of reality. What a slur on his reputation!

Here is the link: Hodges on Copeland/Proudfoot

In 2006, Martin Davis could not stand it any longer, and published a paper

The Myth of Hypercomputation

In the paper, he very sweetly demolishes, annihilates and eviscerates the idea of "hypercomputation." Again, this kind of direct and scathing criticism is highly unusual; bad work is typically ignored, not openly criticized.

Needless to say, the Cult of Hypercomputation simply ignores Davis's paper, as well as all other criticism.

Currently, several areas of what used to be pure, hard science have been invaded by fashionable nonsense and—sadly—even fraud[†]

There is some amusement value in this, and it even may be welcome as a sign of a more open, less dogmatic and hierarchical world. For example, `arXiv` is a big step forward from the traditional publishing machine and some math/cs blogs and videos are absolutely great. The flip-side is `viXra`, an inexhaustible source of garbage, and a lot of material on the web is simply trash.

Democratizing science is great. But, there is a substantial danger, without any control mechanism whatsoever we get "alternative facts" instead of science.

It's a slippery slope from shoddy work to just plain nonsense. And worse.

[†]Recent examples: the room-temperature superconductor fiasco at UoR; 29% of certain biomedical papers irreproducible.

> This drifting of figures and geometric figuring, this irruption of dimensions and transcendental mathematics, leads to the promised surrealist peaks of scientific theory, peaks that culminate in Gödel's theorem: the existential proof, a method that mathematically proves the existence of an object without producing the object.
>
> Paul Virilio

Virilio is described as a "cultural theorist, urbanist, and aesthetic philosopher."