

Deciding Arithmetic in Malament-Hogarth Spacetimes

Mark Hogarth
Department of History and Philosophy of Science
Cambridge
e-mail: mh10026@cam.ac.uk

Submitted to *BJPS*, October 12, 2001

Abstract Presented here are some new results concerning the computational power of so-called SAD_n computers, a class of Turing machine-based computers that utilise the geometry of Malament-Hogarth spacetimes to perform non-Turing computable feats. The main result is that SAD_n can decide n -quantifier arithmetic but not $(n+1)$ -quantifier arithmetic, a result which reveals how neatly SAD_n s map into the Kleene arithmetical hierarchy.

Introduction

Recent investigations have thrown light upon a general relativistic computing device that can perform some non-Turing computable tasks (Hogarth 1992, Earman and Norton 1993, 1994, Etesi and Nemeti 2001). This computer is actually one member of a large family of non-Turing computers (SAD_n s), as described by Hogarth (1994). In this paper I will give some new results concerning the upper and lower computational limits of the power of this family's members.

To begin with, we recall how a Turing machine (TM) is ordinarily employed to solve a mathematical problem. In Figure 1, the computer user, CU , travels alongside the TM waiting for the TM to signal the solution to the problem. The event at which CU receives the solution (if one exists) I will call the *solution event*. Naturally one assumes that there is enough space and time to allow the TM to achieve these ends. This will be satisfied if spacetime is, e.g., Newtonian.

Figure 1 here.

But one can consider TMs operating in other spacetimes, e.g. the models of general relativity (GR), which, as it happens, is currently our best theory of spacetime in the large. A TM operating in Minkowski spacetime, the simplest of GR models, presents a situation analogous to the Newtonian case, except now the *CU* can reach the solution event in an arbitrarily small span of time or rather *proper time*. This follows because in any relativistic spacetime the elapsed proper time between two events depends not merely on the location of the events (as it does in Newtonian spacetime) but on the path taken between those events. The faster *CU* travels, the shorter the proper time taken to reach the solution event.

This, however, does not provide a way to perform non-Turing computable tasks. For that one could consider a class of spacetime dubbed by Earman and Norton (1993), *Malament-Hogarth (M-H)*. In Figure 1, *CU* and TM start together as before, but now *CU* can, in a finite span of proper time, position herself to the *entire* future of the TM's infinitely long (i.e. infinite proper time) worldline. This is possible because M-H spacetimes possess, figuratively speaking, a 'circumventable edge' to spacetime. Though at odds with one's intuitions about space and time, this arena is consistent within the framework of general relativity.

Note that neither TM nor *CU* completes an infinite number of tasks in a finite amount of their proper time, so the scenario does not entail a 'completed infinity'.¹

Another kind of TM-based computer can be constructed within some (but not all) M-H spacetimes. The basic idea is to 'string' together a sequence of 'first-order' computers (SAD_1 : definition coming up) computers to give a second-order computer (SAD_2). Likewise, stringing SAD_2 s together gives a SAD_3 . And so on. Thus we speak of the set of SAD_n computers. Finally stringing together SAD_1 , SAD_2 , SAD_3 ... gives rise to an AD computer. The physics of these computers is investigated by Hogarth (1994) (see also Earman and Norton 1993), but in the next section their logical structure will be revealed by their schematic representations.

From these representations, one will see that none of the SAD_n s entail a 'completed infinity'.

¹For this reason I would argue that 'supertask' is not a word that should be applied to this set-up, although it has been in the literature.

Axiomatising the computers

Textbook Turing computability theory usually begins with an argument aimed at showing that any finite number of ordinary computing machines in any configuration can always be mimicked by a single appropriately programmed TM. The theory then proceeds in terms of this one abstract and easily characterised machine, and thereby manages to transcend irrelevant hardware details. Indeed it is this very process of abstraction that makes Turing computability a branch of pure mathematics.

This approach can be applied to other TM-based computers. This will include the TM that operates in the ordinary way and also a device like a TM except that it exists for only a finite time. I will call the first computer an *ordinary TM* (abbreviation: OTM) and the second a *finite TM* (abbreviation: FTM). The 'axiomatization' of a particular physical computer is achieved by providing a simple yet computationally equivalent schematic representation — a blue print, if you like. These are depicted in figure 2.

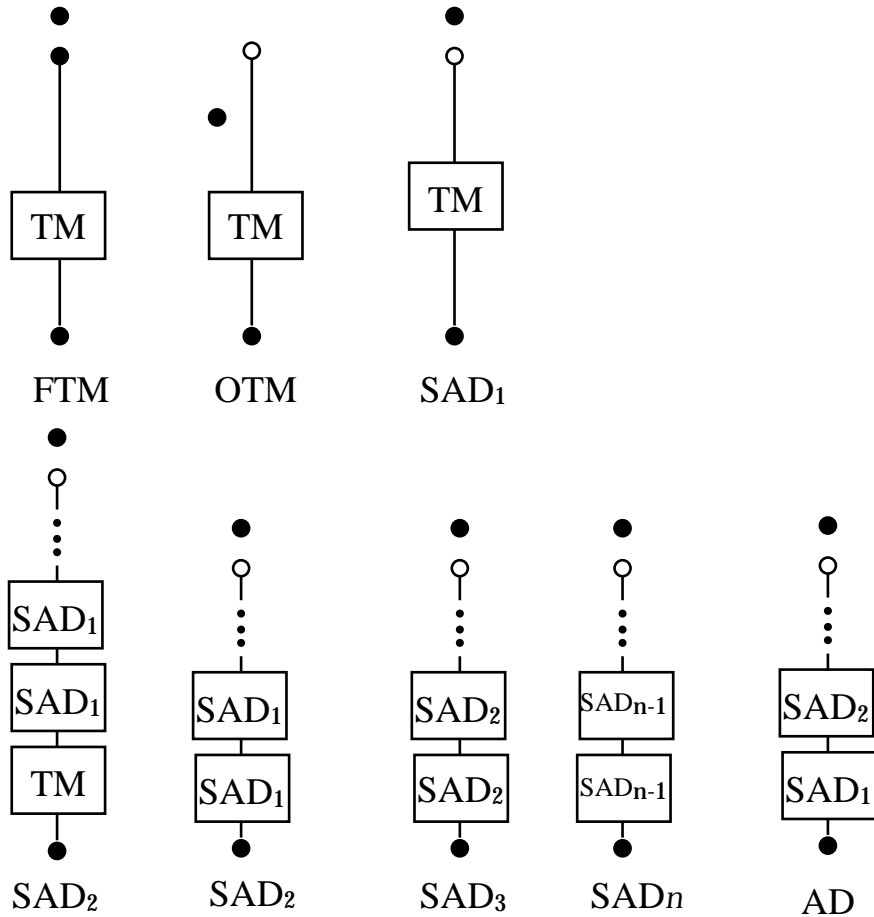


Figure 2. Blue prints of TM-based machines. ‘ ’ means ‘is equivalent to’.

The diagrams are based on spacetime diagrams, but the reader can—and from a pure mathematical viewpoint *should*—view them as purely schematic. A filled dot represents a *solution event*; a line is a hardware worldline; an empty dot is ‘the edge of spacetime at infinity’; three dots means that the sequence of hardware boxes continues indefinitely. In the FTM diagram, there is a single finite TM that travels some finite temporal distance (time’s up), and lies below (in the past) of a particular solution event. Next is the OTM: one TM travels without stopping and the solution event lies ahead of a finite part of the TM. The SAD_1 (short for ‘first order arithmetical sentence deciding’) has one TM travelling without stopping, but this time the whole of TM’s worldline lies below the solution event. SAD_2 is a string without end of SAD_1 computers: each box houses a SAD_1 computer. Two representations are given: the one on the left has a TM at the beginning to remind us where the (finite) program that drives each of the SAD_1 s resides. The representation on the right is used when no such reminder is needed. Last in the figure is the AD (short for ‘arithmetic deciding’) computer: a string of increasingly powerful SAD machines.

To reflect actual computers 4 self-explanatory rules govern the 'transfer of information' within the schemata (this is more fully described in Hogarth 1994).

- (1) a lower box can signal to a higher box or to the solution event;
- (2) each box and the signal event can receive at most one signal (no swamping);
- (3) all boxes receive their instructions from the initial TM;
- (4) downward signalling is forbidden.

The power of the TM-based computers

This will be measured in terms of a computer's capacity to decide part or all of arithmetic in the standard interpretation. Sentences of arithmetic will here be represented by their relational counterparts (see Boolos and Jeffrey 1989, Rogers 1987).

Lemma 1. A SAD_1 can decide 1-quantifier arithmetic.

Proof. Suppose a relation $S = \exists x R(x)$, where R is recursive. If the TM of the SAD_1 runs through $R(1), R(2), \dots$, and signals iff one such sentence is true, then S is decided by SAD_1 . Likewise if $S = \forall x R(x)$.

Lemma 2. Suppose $S(1), S(2), S(3), \dots$ is a sequence of sentences that can be enumerated by a TM. Suppose further that for each $m \geq 1$, $S(m)$ is decidable by SAD_n . Then $\exists m S(m)$ and $\forall m S(m)$ are both decidable by SAD_{n+1} .

Proof. Let $P(m)$ denote the procedure that by hypothesis decides $S(m)$. The procedure for deciding $\exists m S(m)$ consists of having

$P(1)$ signals to q and $P(2)$ if and only if $S(1)$ holds.

For $m > 1$, $P(m)$ signals to $P(m+1)$ if and only if $P(m-1)$ signals to $P(m)$.

For $m > 1$, $P(m)$ signals to q and $P(m+1)$ if and only if $S(m)$ holds and $P(m)$ has not received a signal from $P(m-1)$.

This procedure ensures that a *single* signal is sent to q if and only if there is an m such that $S(m)$ holds. (The signal is actually sent by $P(x)$, where x is the smallest integer

for which $S(x)$ holds.) Upshot: a signal at q means $\neg \exists m S(m)$ holds (true), no signal at q means $\neg \exists m S(m)$ does not hold (false).

The procedure for deciding $\exists m S(m)$, given below, is similar except this time a single signal at q means $\exists m S(m)$ does not hold, no signal at q means $\exists m S(m)$ does hold.

$P(1)$ signals to q and $P(2)$ if and only if $\neg S(1)$ holds.

For $m > 1$, $P(m)$ signals to $P(m+1)$ if and only if $P(m-1)$ signals to $P(m)$.

For $m > 1$, $P(m)$ signals to q and $P(m+1)$ if and only if $\neg S(m)$ holds and $P(m)$ has not received a signal from $P(m-1)$.

Thus $\exists m S(m)$ and $\neg \exists m S(m)$ are seen to be decidable by SAD_{n+1}

We have seen already how single-quantifier sentences can be decided by SAD_1 . Now suppose that we are given an arbitrary double-quantifier sentence $S = Q_1(y)Q_2(z)F(y,z)$, where F is a recursive relation and Q_i is either \forall or \exists . Writing S as $Q_1(y)\{Q_2(z)F(y,z)\}$ means that lemma 2 can be applied (F recursive obviously implies the sentences $Q_2(z)F(1,z), Q_2(z)F(2,z), \dots$ can be listed in order) to show that double-quantifier sentences are decidable by SAD_2 . A further application of the lemma shows that triple-quantifier sentences can be decided in SAD_3 . And continuing in this way we arrive at the following fundamental result.

*Proposition 3.*² n -tuple quantifier arithmetic is decidable by SAD_n .

Moreover, since the AD embodies SAD_n s of every order of n , we have the following

Proposition 4. Arithmetic is decidable by AD.

But the AD machine is not omniscient, as the corollary of the following result shows.

Proposition 5 (Lucian Wischik³). AD can compute exactly \aleph_0 functions.

Proof. Since each AD machine is characterised by the finite program that drives it, the set of AD machines can be given a Gödel-type numbering. The proof now follows the proof of the corresponding OTM result (Rogers 1987, p. 22).

Corollary. There exist functions that the AD machine cannot compute.

² A 'physical' version of this theorem was proved in Hogarth (1994)

³ Private communication.

Proposition 3 is a positive result about the power of SAD_n , but it raises a question: just how far does the power of the SAD_n extend? The case of $n=1$ has been investigated by Earman and Norton (1994), who found that SAD_1 cannot decide arbitrary sentences with two quantifiers. I will now show that this statement generalises to any n : SAD_n cannot decide arbitrary sentences with $n+1$ quantifiers. This is the main theorem. I will first need to rehearse some standard theory (Rogers 1987, chapter 14).

Let Σ_n (respectively Π_n) denote the set of relations expressible by a series of n quantifiers that begin with \exists (respectively \forall) and act on a recursive relation. For example, if the two-place relation $R(u,v) = \exists x \forall y S(x,y,u,v)$, where S is recursive, then $R \in \Sigma_2$. Let Δ_n denote the set of relations that are expressible in both such forms; hence $\Delta_n = \Sigma_n \cap \Pi_n$.

These three results are standard: $\Sigma_n \cap \Pi_n = \Sigma_{n+1} \cap \Pi_{n+1} = \Delta_{n+1}$; $(\Sigma_n \cap \Pi_n) \subset \Sigma_{n+1} \cap \Pi_{n+1}$; $(\Pi_n \cap \Sigma_n) \subset \Pi_{n+1} \cap \Sigma_{n+1}$. It follows that the classes $\Sigma_0, \Sigma_1, \Sigma_2, \Sigma_3, \dots$ and the classes $\Pi_1, \Pi_2, \Pi_3, \dots$ each form a strictly increasing sequence. This result is known the *arithmetical hierarchy theorem*.⁴ It can be interpreted as saying that mathematics becomes genuinely more complex as the number of (alternating) quantifiers increases.

The notion of a computing *oracle* is due to Turing. One can think of it as a 'black box' that possesses by fiat certain well-defined powers of computation (e.g. it can decide only recursively enumerable sets). The corresponding *oracle machine* is a TM that can consult the oracle during any step of its operations. So, for example, by a Π_1 -*oracle-machine* I mean an OTM whose repertoire of ordinary operations is supplemented by the capacity to receive, when required, correct answers to any question of just the form 'does $R(x)$ hold?', where $R \in \Pi_1$.

Oracles and the arithmetical hierarchy come together in this version of *Post's theorem*. A Π_n -oracle machine (Σ_n -oracle machine) can decide relations in precisely Δ_{n+1} . This shows that if one had complete 'access' to a particular level of the arithmetical hierarchy, then one could recursively access no more than a small subset of the next level up.

I will adopt the standard practice of abbreviating relations like $(\exists x)(\forall y)(\exists z)R(u,x,y,z)$ & $(\forall x)S(u,x)$, where R and S are recursive, to $\exists x \forall y \exists z R(u,x,y,z)$ & $\forall x S(u,x)$; that is, I will indicate only the quantifier symbols and logical connectives. Σ_n (respectively, Π_n) will denote an alternating string of n -quantifiers beginning with \exists (respectively, \forall).

⁴ The special case ' $\Sigma_0 = \Pi_1$ ' represents a statement of Gödel's second incompleteness theorem.

Using the *Tarski-Kuratowski algorithm* any such relation expression involving quantifiers and logical connectives can be reduced to an equivalent series of alternating quantifiers, which in turn allows one to establish where a given relation is located in the arithmetical hierarchy. E.g. \neg is equivalent to Σ_1 ; $\neg(\&)$ is equivalent to Σ_1 , and to Π_1 , and finally to Σ_1 (*ibid.*, p. 309).

Figure 3. Here.

We are now in a position to state the main result.

Proposition 6. SAD_n cannot decide an arbitrary Π_{n+1} sentence (or an arbitrary Σ_{n+1} sentence).

It is helpful first to prove the following.

Lemma 7. Suppose $S(z)$ is 1-place relation that is SAD_n decidable. Then $S(z)$ is of the form $S_1(z) \vee S_2(z)$, where $S_1(z) \in \Pi_n$ and $S_2(z) \in \Sigma_n$.

The proof is by induction.

Case of $n=1$. Following Earman and Norton (1994), if $S(z)$ is decidable by SAD_1 then it must perform in one of four ways:

- (1) for all z , there is a signal (to arrive at the solution event);
- (2) for all z , no signal means that $S(z)$ does not hold;
- (3) for all z , no signal means that $S(z)$ holds;
- (4) for some z no signal means $S(z)$ does not hold, and for some z no signal means $S(z)$ holds.

(In case (4) it must be a Turing computable matter which z means 'signal implies does not hold' and which z means the opposite.)

Case (1) implies that the TM program must halt for each z , so $S(z)$ must take the form $S(z)=R_a(z)$, where R_a is a recursive relation. Thus $S(z) \in \Pi_0$.

Case (2) implies that when $S(z)$ holds, the TM halts after a finite number of steps. This means that for each z a number is reached which halts the TM program. Thus $S(z)$ is expressible as $S(z)=\exists y R_b(y,z)$ for some recursive relation R_b ; i.e. $S(z) \in \Sigma_1$.

Case (3) implies that the TM will halt only if $S(z)$ does not hold, that is, when $\neg S(z)$ does hold. From case (2), this implies that $\neg S(z)$ is Σ_1 , i.e. $S(z)$ is Σ_1 .

Case (4) implies that for some z $S(z)$ is as in case (2), so S is Σ_1 ; and for some z $S(z)$ is as in case (3), so $S(z)$ is Σ_1 . Thus $S(z)$ is Σ_1 .

Collecting together cases (1) through (4) proves the case $n=1$.

The general case. Now suppose the lemma is true for some arbitrary but fixed m , i.e. if $S(z)$ is SAD_m -decidable, then $S(z)$ is a (one-place) relation of the form $S_1(z) \vee S_2(z)$, where $S_1(z) \in \Pi_m$ and $S_2(z) \in \Sigma_m$.

Consider the SAD_{m+1} schematic in figure 3. It consists of a string of SAD_m components. Relations that a SAD_m component can decide can, by the hypothesis of the previous paragraph, be decided by a Π_m -oracle. Thus by replacing each SAD_m component by a Π_m -oracle, there is no loss of computational power. Call this machine X . The question now is: what is the power of X ?

Suppose $T(z)$ is a 1-place relation that is decidable by X . Then, as above, X must perform in one of four ways:

- (1) for all z , there is a signal;
- (2) for all z , no signal means that $T(z)$ does not hold;
- (3) for all z , no signal means that $T(z)$ does hold;
- (4) for some z no signal means $T(z)$ does not hold, and for some z no signal means $T(z)$ does hold.

By analogy with $n=1$ case and applying Post's theorem, we see that:

Case (1) implies that $T(z) \in \Delta_{m+1}$.

Case (2) implies that $T(z) = \exists y R_b(y, z)$, where $R_b(y, z) \in \Delta_{m+1}$. This implies that $T(z)$ is Σ_{m+1} and $T(z)$ is Σ_{m+1} , so by the Tarski-Kuratowski algorithm $T(z)$ is Σ_{m+1} ,

Case (3) implies that $T(z)$ is Σ_{m+1} and $T(z)$ is Σ_{m+1} , so by the Tarski-Kuratowski algorithm $T(z)$ is Σ_{m+1} .

Case (4) implies that $T(z)$ is $\Sigma_{m+1} \cup \Sigma_{m+1}$.

Collecting together cases (1) through (4) shows that $T(z)$ Σ_1 -decidable implies $T(z)$ is Σ_{m+1} -decidable. But we know that $T(z)$ Σ_{m+1} -decidable implies $T(z)$ Σ_1 -decidable. Thus $T(z)$ Σ_{m+1} -decidable implies $T(z)$ is Σ_{m+1} -decidable.

Hence the m th case implies the $(m+1)$ th case. Since the $m=1$ holds, the proof is complete.

Corollary. $S(z) \in \Delta_{n+1}$.

Proof. Apply the Tarski-Kuratowski algorithm.

Proof of proposition 6. By the Kleene hierarchy theorem there are relations in Π_{n+1} that are not in Δ_{n+1} . Thus if $U(z)$ is one such 1-place relation then the corresponding family of sentences cannot be decidable by Σ_n . The same applies to Σ_{n+1} .

Combining propositions 3 and 6 gives:

Proposition 8. Σ_n can decide Π_n Σ_n but not Π_{n+1} Σ_{n+1} .

This shows how neatly Σ_n s map into the Kleene arithmetical hierarchy.

Closing remarks

A Σ_n can decide n -tuple but not $(n+1)$ -tuple arithmetic. The AD can decide arithmetic, but is not omniscient.

The concept of computability arguably takes on a new appearance in the light of these results. In Turing's picture it seemed as if there was a fundamental 'boundary' that separates decidable problems from undecidable problems. In the new picture, however, this boundary is seen to lie within a series of decidable/undecidable boundaries, each corresponding to the power of a different computer. In short, decidability is relativised. Arithmetic — to take one celebrated problem — is OTM-undecidable (Gödel showed us that) and AD-decidable.

References

- Boolos, G.S. and Jeffrey, R.C. (1989). *Computability and Logic*. Cambridge: Cambridge University Press.
- Earman, J. (1995). *Bangs Crunches Whimpers and Shrieks: Singularities and Acausalities in Relativistic Spacetimes*. Oxford: Oxford University Press.
- Earman, J. and Norton, J. (1993). 'Forever is a Day: Supertasks in Pitowsky and Malament-Hogarth Spacetimes', *Philosophy of Science*, 5, 22-42.

Earman, J and Norton, J. (1994). 'Infinite Pains: The Trouble with Supertasks', to appear in S. Stich (ed), *Paul Benacerraf: The Philosopher and His Critics*. New York: Blackwell.

Etesi and Nemeti (2001) Preprint.

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', in D. Hull, M. Forbes, and R. M. Burian (eds), *PSA 1994*, Vol. 1. East Lansing: Philosophy of Science Association, 126-138.

Rogers, H. (1987). *Theory of Recursive Functions and Effective Computability*. Cambridge, MA: MIT Press.

Wald, R. M. (1984). *General Relativity*. Chicago: University of Chicago Press.