Direct link to author's version

INCOMPLETENESS FOR STABLY SOUND TURING MACHINES

YASHA SAVELYEV

Abstract. We first partly develop a mathematical notion of stable soundness intended to reflect the actual soundness property of human beings. Then we show that given an abstract query machine M the following cannot hold simultaneously: M is stably sound, M is computable, M can stably decide the truth of any arithmetic statement. This can be understood as an extension of the Gödel incompleteness theorem to stably sound setting. This is a non-trivial extension as a stably sound Turing machine can decide the halting problem. In practice such an M could be meant to represent a weakly idealized human being so that the above gives an obstruction to computability of intelligence, and this gives a formal extension of a famous disjunction of Gödel.

1. Introduction

The term ***machine*** [1] will just be a synonym for a partial map $M : A \to B$, with A, B sets with an additional structure of an encoding in $\mathbb{N}$. An encoding is just an injective map $e : A \to \mathbb{N}$ with some extra properties. This is described in more detail in Section 2.2.

Let $\mathcal{A}$ denote the set of (first order) sentences of arithmetic. And suppose we are given a machine

$$M : \mathbb{N} \to \mathcal{A} \times \{\pm\},$$

for $\{\pm\}$ a set with two elements $+, -$. If there is an n_0 with $M(n_0) = (\Sigma, +)$ s.t. there is no $m > n_0$ with $M(m) = (\Sigma, -)$ then we say that Σ ***is*** M***-stable***.

Definition 1.1. *We say that a machine*

$$M : \mathbb{N} \to \mathcal{A} \times \{\pm\}$$

is **stably sound** *if for all* $\Sigma \in \mathcal{A}$*:*

$$\Sigma \text{ is } M\text{-stable} \implies \Sigma \text{ is true.}$$

We say that M **decides arithmetic** *if*

$$\forall \Sigma \in \mathcal{A} : \Sigma \text{ is true} \implies \Sigma \text{ is } M\text{-stable.}$$

So such an M has the following interpretation:

$$M(n) = (\Sigma, +)$$

if at the moment n M decides that Σ is true, while

$$M(m) = (\Sigma, -)$$

if at the moment m M does not decide that Σ is true. If Σ is M-stable we can also say M ***stably decides*** Σ to be true. So that M is stably sound if and only if whenever M stably decides Σ to be true, it is in fact true.

The following is one version of our main theorem, with Theorem 4.7 being a more fundamental variant. We present this formulation first because a human mathematician intuitively works as a machine M above.

Theorem 1.2. *For M as above the following cannot hold simultaneously: M is stably sound, M is computable and M decides arithmetic. Here 'computable' has the mostly standard meaning of computability by a Turing machine, with specifics given in Section 2.2.*

[1] For some authors and in some of the writing of Turing and Gödel "machine" is synonymous with Turing machine. For us the term machine is just abstraction for a process acting on inputs, but it need not be a computational process, in contrast to Turing machines.

If we replace stably sound by sound then the above would just be a reformulation of Gödel's incompleteness theorem. Moreover, if we specialize our argument to this case then in the statement of the theorem instead of M deciding arithmetic we only need M to decide sentences (in Turing machine language) of the kind: the Turing machine T does not halt on input n. In other words, we would get the well known result of Turing that the halting problem is not soundly decidable by a Turing machine. On the other hand there is a stably sound Turing machine which can stably soundly decide the halting problem, cf. Example 3.3. So perhaps Turing machines stably soundly decide the truth of any arithmetic statement? The above theorem shows that this is not the case.

1.1. **Motivational background - intelligence, Gödel's disjunction and Penrose.** In what follows we understand *human intelligence* very much like Turing in [2], as a black box which receives inputs and produces outputs. More specifically, this black box M is meant to be some system which contains a human subject. We do not care about what is happening inside M. So we are not directly concerned here with such intangible things as understanding, intuition, consciousness - the inner workings of human intelligence that are supposed as special. The only thing that concerns us is what output M produces given an input, not how it is produced. Given this *very* limited interpretation, the question that we are interested in is this:

Question 1. Can human intelligence be completely modelled by a Turing machine?

An informal definition of a Turing machine (see [1]) is as follows: it is an abstract machine which permits certain inputs, and produces outputs. The outputs are determined from the inputs by a fixed finite algorithm, defined in a certain precise sense. For a non-expert reader we point out that this "fixed" does not preclude the corresponding machine from "learning", [2] it just means that how it "learns" is completely determined by the initial algorithm. In particular anything that can be computed by computers as we know them can be computed by a Turing machine. For our purposes the reader may simply understand a Turing machine as a digital computer with unbounded memory running some particular program. Unbounded memory is just a mathematical convenience. In specific arguments, also of the kind we make, we can work with non-explicitly bounded memory. Turing himself has started on a form of Question 1 in his "Computing machines and Intelligence" [2], where he also informally outlined a possible obstruction to a yes answer coming from Gödel's incompleteness theorem.

For the incompleteness theorem to have any relevance we need some assumption on the soundness or consistency of human reasoning. Informally, a human is sound if whenever they asserts something in absolute faith this something is indeed true. This requires context as truth in general is undefinable. For our arguments later on the context will be in certain mathematical models. However, we cannot honestly hope for soundness, as even mathematicians are not on the surface sound at all times, they may assert mathematical untruths at various times, (but usually not in absolute faith). But we can certainly hope for some kind of fundamental soundness.

In this work we will formally interpret fundamental soundness as stable soundness, which we already partly described above. This notion of stable soundness is meant to reflect the basic understanding of the way science progresses. Of course even stable soundness needs idealizations to make sense for individual humans. The human brain deteriorates and eventually fails, so that either we idealize the human brain to never deteriorate in particular never die, or M now refers not to an individual human but to the evolving scientific community. We call such a human ***weakly idealized***.

Around the same time as Turing, Gödel argued for a no answer to Question 1, see [11, 310], relating the question to existence of absolutely unsolvable Diophantine problems, see also Feferman [6], and Koellner [13], [14] for a discussion. Essentially, Gödel argues for a disjunction:

$$\neg((S \text{ is computable}) \wedge (S \text{ is sound}) \wedge A),$$

where S refers to a certain idealized subject, and where A says that S can decide any Diophantine problem. Gödel's argument can be formalized, see [14]. At the same time Gödel doubted that $\neg A$ is

[2] In the sense of "machine learning".

possible, again for an idealized S, as this puts absolute limits on what is humanly knowable even in arithmetic. Note that his own incompleteness theorem only puts relative limits on what is humanly knowable, within a fixed formal system.

However, what is the meaning of 'idealized' above? If idealized just means stabilized in the sense of this paper (Section 3 specifically) then there is a Turing machine T whose stabilization T^s provably decides the halting problem, cf. Example 3.3, and so T^s is no longer computable. In that case, the above disjunction becomes meaningless because passing to the idealization may introduce non-computability where there was none before. So in this context one must be extremely detailed with what "idealized" means physically. The process of the physical idealization must be such that non-computability is not introduced in the ideal limit. In the case of weak idealization mentioned above, it should certainly be in principle possible but this not what is needed by Gödel. He needs an idealization that is plausibly sound otherwise the disjunction would again be meaningless, while a weak idealization of a human is only plausibly stably sound. Since we have no idea what is happening in the human brain, it is not at all clear that what Gödel asks is even possible.

So the natural solution to attempt is to enrich the argument of Gödel so that it explicitly allows for just stable soundness. But then we may worry: if stable soundness is such a loose concept that a Turing machine machine can stably soundly decide the halting problem, maybe Turing machines can stably soundly decide anything? So we need a new incompleteness theorem and this is the theorem above.

After Gödel, Lucas [10] and later again and more robustly Penrose [16] argued for a no answer based only on soundness and the Gödel incompleteness theorem, that is attempting to remove the necessity to decide A or $\neg A$. A number of authors, particularly Koellner [13], [14], argue that there are likely unresolvable meta-logical issues with the Penrose argument, even allowing for soundness. See also Penrose [16], and Chalmers [4] for discussions of some issues. The issue, as I see it, is loosely speaking the following. The kind of argument that Penrose proposes is a meta-algorithm P that takes as input specification of a Turing machine or a formal system, and has as output a natural number (or a string, sentence). Moreover, each step of this meta-algorithm is computably constructive. But the goal of the meta-algorithm P is to prove P is not computable as a function! So on this rough surface level this appears to be impossible. But it may be possible to modify the idea in some subtle way.

Notwithstanding, what we argue here is that there is much more compelling version of the original Gödel disjunction that only needs stable soundness. The following is a slightly informal, applied version of our Theorem 1.2.

Theorem 1.3. *Either there are cognitively meaningful, absolutely non Turing computable processes in the human brain, or (weakly idealized) human beings are stably unsound, or for any S there exists a certain true arithmetic statement, let's call it $\mathcal{H}(S)$ [3], that S will never stably decide to be true. The above is indeed a mathematical fact [4], given our formalizations.*

By *absolutely* we mean in any sufficiently accurate physical model. Note that even existence of absolutely non Turing computable processes in nature is not known. For example, we expect beyond reasonable doubt that solutions of fluid flow or N-body problems are generally non Turing computable (over $\mathbb{Z}$, if not over $\mathbb{R}$ cf. [3]) [5] as modeled in essentially classical mechanics. But in a more physically accurate and fundamental model they may both become computable, possibly if the nature of the universe is ultimately discreet. It would be good to compare this theorem with Deutch [5], where computability of any suitably finite and discreet physical system is conjectured. Although this is not immediately at odds with us, as the hypothesis of that conjecture may certainly not be satisfiable.

Remark 1.4. *It should also be noted that for Penrose, in particular, non-computability of intelligence would be evidence for new physics, and he has specific and* very *intriguing proposals with Hameroff* [9]

[3] $\mathcal{H}(S)$ is a statement in the language of Turing machines and so is number theoretic, however it is not a Diophantine problem. Of course it cannot be by Example 3.3.

[4] Specifically a theorem of set theory, although we keep set theory implicit as usual.

[5] We are now involving real numbers but there is a standard way of talking of computability in this case, in terms of computable real numbers. This is what means over $\mathbb{Z}$.

on how this can take place in the human brain. As we have already indicated, new physics is not a logical necessity for non-computability of brain processes, at least given the state of the art. However, it is very plausible that new physical-mathematical ideas may be necessary to resolve the deep mystery of human consciousness. Here is also a partial list of some partially related work on mathematical models of brain activity, consciousness and or quantum collapse models: [12], [15], [7], [8].

As a final remark, technically the paper is mostly elementary and should be widely readable in entirety.

2. Some preliminaries

This section can be just skimmed on a first reading. Really what we are interested in is not Turing machines per se, but computations that can be simulated by Turing machine computations. These can for example be computations that a mathematician performs with paper and pencil, and indeed is the original motivation for Turing's specific model. However to introduce Turing computations we need Turing machines. Here is our version, which is a computationally equivalent, minor variation of Turing's original machine.

Definition 2.1. *A* **Turing machine** M *consists of:*

- *Three infinite (1-dimensional) tapes T_i, T_o, T_c, (input, output and computation) divided into discreet cells, next to each other. Each cell contains a symbol from some finite alphabet Γ with at least two elements. A special symbol $b \in \Gamma$ for blank, (the only symbol which may appear infinitely many often).*
- *Three heads H_i, H_o, H_c (pointing devices), H_i can read each cell in T_i to which it points, H_o, H_c can read/write each cell in T_o, T_c to which they point. The heads can then move left or right on the tape.*
- *A set of internal states Q, among these is "start" state q_0. And a non-empty set $F \subset Q$ of final states.*
- *Input string Σ: the collection of symbols on the tape T_i, so that to the left and right of Σ there are only symbols b. We assume that in state q_0 H_i points to the beginning of the input string, and that the T_c, T_o have only b symbols.*
- *A finite set of instructions: I, that given the state q the machine is in currently, and given the symbols the heads are pointing to, tells M to do the following. The actions taken, 1-3 below, will be (jointly) called an* **executed instruction set** *or just* **step***:*
  1. *Replace symbols with another symbol in the cells to which the heads H_c, H_o point (or leave them).*
  2. *Move each head H_i, H_c, H_o left, right, or leave it in place, (independently).*
  3. *Change state q to another state or keep it.*
- *Output string Σ_{out}, the collection of symbols on the tape T_o, so that to the left and right of Σ_{out} there are only symbols b, when the machine state is final. When the internal state is one of the final states we ask that the instructions are to do nothing, so that these are frozen states.*

Definition 2.2. *A* **complete configuration** *of a Turing machine M or* **total state** *is the collection of all current symbols on the tapes, position of the heads, and current internal state. Given a total state s, $\delta(s)$ will denote the successor state of s, obtained by executing the instructions set of M on s, or in other words $\delta(s)$ is one step forward from s.*

So a Turing machine determines a special kind of function:

$$\delta^M : \mathcal{C}(M) \to \mathcal{C}(M),$$

where $\mathcal{C}(M)$ is the set of possible total states of M.

Definition 2.3. *A* **Turing computation**, *or* **computation sequence** *for M is a possibly not eventually constant sequence*

$$*M(\Sigma) := \{s_i\}_{i=0}^{i=\infty}$$

of total states of M, determined by the input Σ and M, with s_0 the initial configuration whose internal state is q_0, and where $s_{i+1} = \delta(s_i)$. If elements of $\{s_i\}_{i=0}^{i=\infty}$ are eventually in some final machine state, so that the sequence is eventually constant, then we say that the computation **halts**. *For a given Turing computation $*M(\Sigma)$, we will write*

$$*M(\Sigma) \to x,$$

if $*M(\Sigma)$ halts and x is the corresponding output string.

We write $M(\Sigma)$ for the output string of M, given the input string Σ, if the associated Turing computation $*M(\Sigma)$ halts. Denote by $Strings$ the set of all finite strings of symbols in Γ, including the empty string ϵ. Then a Turing machine M determines a partial function that is defined on all $\Sigma \in Strings$ s.t. $*M(\Sigma)$ halts, by $\Sigma \mapsto M(\Sigma)$.

In practice we will allow our Turing machine T to reject some elements of $Strings$ as valid input. We may formalize this by asking that there is a special final machine state q_{reject} so that $T(\Sigma)$ halts with q_{reject} for

$$\Sigma \notin \mathcal{I} \subset Strings.$$

The set $\mathcal{I}$ is also called the set of T-***permissible*** input strings. We do not ask that for $\Sigma \in \mathcal{I}$ $*T(\Sigma)$ halts. If $*T(\Sigma)$ does halt then we will say that Σ is T-***acceptable***.

Definition 2.4. *We denote by $\mathcal{T}$ the set of all Turing machines with a distinguished final machine state q_{reject}.*

It will be convenient to forget q_{reject} and instead write

$$T : \mathcal{I} \to \mathcal{O},$$

where $\mathcal{I} \subset Strings$ is understood as the subset of all T-permissible strings, or just ***input set*** and $\mathcal{O}$ is the set output strings or ***output set***.

Definition 2.5. *Given a partial function*

$$f : \mathcal{I} \to \mathcal{O},$$

we say that a Turing machine in $T \in \mathcal{T}$

$$T : \mathcal{I} \to \mathcal{O}$$

computes *f if $T = f$ as partial functions on $\mathcal{I}$.*

2.1. **Multi-input Turing machine.** There is a basic well known variant of a Turing machine which takes as input an element of $Strings^n$, for some fixed $n \in \mathbb{N}$. This is done by replacing the input 1-tape by an n-tape. We will not give details of this. Notationally such a Turing machine will be distinguished by a superscript so

$$T^n : Strings^n \to Strings,$$

denotes such a Turing machine with n inputs.

2.2. **Abstractly encoded sets.** We will sometimes use abstract sets to refer to input and output sets. However, these are understood to be subsets of $Strings$ under some implicit, *fixed* encoding. Concretely an ***encoding*** of A is an injective set map $e : A \to Strings$. For example we may encode $Strings^2$ as a subset of $Strings$ as follows. The encoding string of $\Sigma = (\Sigma_1, \Sigma_2) \in Strings^2$ will be of the type: "this string encodes an element $Strings^2$: its components are Σ_1 and Σ_2." In particular the sets of integers $\mathbb{N}, \mathbb{Z}$, which we may use, will under some encoding correspond to subsets of $Strings$. Indeed this abstracting of sets from their encoding in $Strings$ is partly what computer languages do.

More formally, let $\mathcal{S}$ be an arrow category whose objects are maps $e_A : A \to Strings$, for e_A an embedding called ***encoding map of*** A, determined by $A \in Set$. We may denote by $e_A(A)$ by A_e.

The morphisms in the category $\mathcal{S}$ are pairs of partial maps $m = (m_e, m)$ so that the following diagram commutes:

$$\begin{array}{ccc} A & \xrightarrow{\quad m \quad} & B \\ \downarrow{\scriptstyle e_A} & & \downarrow{\scriptstyle e_B} \\ A_e \subset Strings & \xrightarrow{m_e} & B_e \subset Strings, \end{array}$$

and so that m_e is computable and A_e coincides with the set of permissible strings for the corresponding Turing machine. We may just write $A \in \mathcal{S}$ for an object, with e_A implicit. We call such an A an ***abstractly encoded set*** so that $\mathcal{S}$ is a category of abstractly encoded sets.

In addition we ask that $\mathcal{S}$ satisfies the following properties.

(1) For $A \in \mathcal{S}$ A_e is computable (recursive). Here, as is standard, a set $S \subset Strings$ is called *computable* if both S and its complement are computably enumerable, with S called *computably enumerable* if there is a Turing machine T so that $*T(\Sigma)$ halts iff $\Sigma \in S$.

(2) There is an abstractly encoded set $\mathcal{U} = Strings \in \mathcal{S}$, with $e_{\mathcal{U}} = id_{Strings}$. We can think of $\mathcal{U}$ as the set of typeless strings.

(3) For $A, B \in \mathcal{S}$,

$$(e_A(A) \cap e_B(B) \text{ is non-empty}) \implies (A = \mathcal{U}) \vee (B = \mathcal{U}).$$

In particular each $A \in \mathcal{S}$ is determined by the image $e_A(A)$.

(4) If $A, B \in \mathcal{S}$ then $A \times B \in \mathcal{S}$ and the projection maps $pr^A : A \times B \to A$, $pr^B : A \times B \to B$ complete to morphisms of $\mathcal{S}$, similar to the above, so that we have a commutative diagram:

$$\begin{array}{ccc} A \times B & \xrightarrow{pr^A} & A \\ \downarrow & & \downarrow \\ (A \times B)_e & \xrightarrow{pr^A_e} & A_e, \end{array}$$

with pr^A_e computable, similarly for pr^B. In addition we ask for the following naturality property. Let f be the composition

$$A_e \times B_e \xrightarrow{(e_A^{-1}, e_B^{-1})} A \times B \xrightarrow{e_{A\times B}} (A \times B)_e,$$

then f is computable meaning that there is a 2-input Turing machine T^2, with $Dom = A_e \times B_e$ the set of T^2-permissible inputs, such that

$$T^2(\Sigma_1, \Sigma_2) = f(\Sigma_1, \Sigma_2)$$

for all (Σ_1, Σ_2) in Dom.

The above axioms suffice for our purposes but there are a number of possible extensions. The specific such category $\mathcal{S}$ that we need will be clear from context later on. We only need to encode finitely many types of specific sets. For example $\mathcal{S}$ should contain an abstract encoding of $\mathbb{Z}, \mathbb{N}, \{\pm\}, \mathcal{T}$, with $\{\pm\}$ a set with two elements $+, -$. The encodings of $\mathbb{N}, \mathbb{Z}$ should be suitably natural so that for example there is a computable total map $S : e(\mathbb{N}) \to e(\mathbb{N})$ which corresponds under e to the self-map $n \mapsto n+1$ of $\mathbb{N}$, and so that there is a bijective morphism $\mathbb{N} \to \mathbb{Z}$ in $\mathcal{S}$.

For $A, B \in \mathcal{S}$, given a partial set map $f : A \to B$ we define

$$f_e := e_B \circ f \circ e_A^{-1},$$

called ***the encoding of*** f.

Definition 2.6. *An abstract Turing machine is kind of generalized morphism of $\mathcal{S}$. Specifically, for $A, B \in \mathcal{S}$ an* **abstract Turing machine**

$$T : Dom \subset A \to B$$

is a subset $Dom \subset A$ for $A \in \mathcal{S}$, a partial map

$$f : Dom \to B$$

and $T_e \in \mathcal{T}$*,*

$$T_e : e_A(Dom) \to e_B(B)$$

such that T_e *computes* f_e*, with* $e_A(Dom)$ *coinciding with the set of* T_e*-permissible strings. The set* Dom *is called the set of* T**-permissible strings**. *We often omit* Dom *from notation, especially when* $Dom = A$*. In practice the partial map* f *may just be denoted by* T *itself.*

We define $\mathcal{T}_{\mathcal{S}}$ to be the set of abstract Turing machines relative to $\mathcal{S}$ as above. We will not need to encode $\mathcal{T}_{\mathcal{S}}$, as there is a natural embedding $i : \mathcal{T}_{\mathcal{S}} \to \mathcal{T}$, $i(T) = T_e$ and $\mathcal{T}$ is encoded, and this will be sufficient for us.

For writing purposes we condense the above as follows.

Definition 2.7. *A* **machine** *will be a synonym for a partial map* $M : Dom \subset A \to B$*, with* A, B *abstractly encoded sets.* Dom *may be omitted from notation. Note* M *is partial on* Dom*, not partial on* A *with* Dom *the set where* M *is defined, so* $Dom \subset A$ *is an extra specified structure. That said we may omit to write* Dom*, in particular when* $Dom = A$*.*

$\mathcal{M} = \mathcal{M}_{\mathcal{S}}$ will denote the set of machines. Given an abstract Turing machine $T : A \to B$, we have an associated machine $fog(T) : A \to B$ defined by forgetting the additional structure T_e. However we may also just write T for this machine by abuse of notation. So we have a forgetful map

$$fog : \mathcal{T} \to \mathcal{M},$$

which forgets the extra structure of a Turing machine.

Definition 2.8. *We say that an abstract Turing machine* T **computes** $M \in \mathcal{M}$ *if* $fog(T) = M$*. We say that* M *is* **computable** *if some* T *computes* M*.*

This has a number of expected properties, for example:

Lemma 2.9. *If* $M : C \to A$ *is computable and* $G : C \to B$ *is computable then*

$$D : C \to A \times B, \quad D(c) = (M(c), G(c))$$

is computable.

Proof. The abstract Turing machines computing M, G will be denoted by M, G still, and let M_e, G_e be the Turing machines as in the definition of abstract Turing machines. We construct a Turing machine D_e via the following meta-algorithm.

(1) Input $\Sigma \in Strings$.
(2) Compute and store the value $M_e(\Sigma)$, Σ may be rejected by M_e in which case D_e rejects Σ.
(3) Compute and store the value $G_e(\Sigma)$, likewise Σ may be rejected by M_e.
(4) Compute and output the value

$$D_e(\Sigma) := T^2(M_e(\Sigma), G_e(\Sigma)),$$

where T^2 is as in the Axiom 4.

At any step the corresponding computation may of course not halt, which just means that our Turing machine D_e does not halt, but notwithstanding D_e is a Turing machine that computes $e_{A\times B} \circ D \circ e_C^{-1}$ by construction. □

Composition. Given Turing machines

$$M_1 : A \to B, M_2 : C \to D,$$

we may naturally ***compose*** them to get a Turing machine $M_2 \circ M_1 : R \to D$, for $R = M_1^{-1}(B \cap C)$, with the intersection taken in $Strings$. R can be empty in which case $M_2 \circ M_1$ is a Turing machine which rejects all input. In the case of abstract Turing machines we ask that $B = C$ and then the composition operation is likewise readily defined. Let us not elaborate further.

2.3. **Notation.** $\mathbb{Z}$ always denotes the set of all integers and $\mathbb{N}$ non-negative integers. We will sometimes specify an (abstract) Turing machine simply by specifying a map

$$T : \mathcal{I} \to \mathcal{O},$$

with the full data of the underlying Turing machine being implicitly specified, in a way that should be clear from context.

3. On stable soundness

Definition 3.1. *Given a machine:*

$$M : \mathbb{N} \to B \times \{\pm\},$$

if there is an n_0 with $M(n_0) = (b, +)$ s.t. there is no $m > n_0$ with $M(m) = (b, -)$ then b is called M**-stable** *and we say that M* **prints** *b* **stably**.

Definition 3.2. *Given a machine*

$$M : \mathbb{N} \to B \times \{\pm\},$$

we define

$$M^s : \mathbb{N} \to B$$

to be the machine enumerating, in order, all the M-stable b. We call this the **stabilization** *of M. The range of M^s is called the* **stable output** *of M.*

In general M^s may not be computable even if M is computable. Explicit examples of this sort can be constructed by hand.

Example 3.3. We can construct a Turing machine

$$A : \mathbb{N} \to \mathcal{P} \times \{\pm\},$$

whose stabilization A^s enumerates every Diophantine (integer coefficients) polynomial with no integer roots, where $\mathcal{P}$ denotes the set of all Diophantine polynomials, (also abstractly encoded). Similarly, we can construct a Turing machine D whose stabilization enumerates pairs (T, n) for $T : \mathbb{N} \to \mathbb{N}$ a Turing machine and $n \in \mathbb{N}$ such that $*T(n)$ does not halt. In other words D stably soundly decides the halting problem. To do this we may proceed via a zig-zag algorithm.

In the case of Diophantine polynomials, here is a (inefficient) example. Let Z computably enumerate every Diophantine polynomial, and let N computably enumerate the integers. In other words, in our language, $Z : \mathbb{N} \to \mathcal{P}$, $N : \mathbb{N} \to \mathbb{Z}$ are total bijective abstract Turing machines.

- Initialize an ordered list L by $L = \emptyset$, which we understand as a list of instructions.
- Start. For each $p \in \{Z(0), \dots, Z(n)\}$ check if one of $\{N(0), \dots, N(n)\}$ is a solution of p, if no add $(p, +)$ to L, if yes add $(p, -)$. Call the resulting list L^{n+1}. Explicitly,

$$L^{n+1} = L \cup \bigcup_{m=0}^{n} (p, d^n(Z(m))),$$
$$\text{where } d^n(p) = + \text{ if none of } \{N(0), \dots, N(n)\} \text{ are roots of } p,\ d^n(p) = - \text{ otherwise,}$$

  where $\cup$ is set union operation of subsets of $\mathcal{P} \times \{\pm\}$.
- Set $L := L^{n+1}$, an ordered list with order starting at 0.
- Set $n := n + 1$ go to Start and continue.

This will define a function $A : \mathbb{N} \to \mathcal{P} \times \{\pm\}$ whose value $A(m)$ is the m'th, not necessarily final, instruction in the list L^{m+1}. Clearly A is computable and it's stabilization A^s enumerates Diophantine polynomials which have no integer roots.

3.1. **Decision machines.** By a ***decision machine*** we mean a machine of the form:

$$D : Dom \subset B \times \mathbb{N} \to \{\pm\}.$$

Definition 3.4. *For a decision machine D we say that $b \in B$ is D-**decided** if there is an $n_0 \in \mathbb{N}$ with $D(b, n_0) = +$ s.t. there is no $m > n_0$ with $D(b, m) = -$.*

Given a machine $M : \mathbb{N} \to B \times \{\pm\}$, there is an associated decision machine:

$$D_M : Dom \subset B \times \mathbb{N} \to \{\pm\},$$

with Dom defined to be the set of $(b, n) \in B \times \mathbb{N}$ such that either $pr_B \circ M(n) = b$ or $M(n)$ is undefined, for $pr_B : B \times \{\pm\} \to B$ the projection. D_M is defined on Dom by

$$D_M(b, n) = \begin{cases} + & \text{if } M(n) = (b, +) \\ - & \text{if } M(n) = (b, -) \\ \text{undefined} & \text{if } M(n) \text{ is undefined.} \end{cases}$$

Lemma 3.5. *If M as above is computable then D_M is computable. Moreover, b is M-stable iff b is D_M-decided.*

Proof. To keep notation simple let M denote the Turing machine computing the machine M. Let

$$\widetilde{M} : B \times \mathbb{N} \to (B \times \{\pm\}) \times B$$

be the Turing machine defined (as a partial map) by

$$\widetilde{M}(b, n) = (M(n), pr^B(b, n) = b).$$

Using Lemma 2.9 and that the projection

$$pr^B : B \times \mathbb{N} \to B$$

is computable by Axiom 4 we see that $\widetilde{M}$ is indeed computable.

Define T to be the composition of (abstract) Turing machines:

$$T := R \circ \widetilde{M},$$

for

$$R : (B \times \{\pm\}) \times B \to \pm,$$

the Turing machine so that $(b, \pm, a)$ is R-rejected, is not R-permitted, whenever $b \neq a$ (we implicitly use Axiom 4 again to computably decide if $b \neq a$). And otherwise

$$R(b, \pm, a) = pr_{\pm}(b, \pm, a)$$

for

$$pr_{\pm} : (B \times \{\pm\}) \times B \to \pm$$

the projection. Then clearly T computes D_M. The second part of the lemma is immediate. □

In particular by the example above there is a Turing machine

$$D_A : \mathcal{P} \times \mathbb{N} \to \{\pm\}$$

that stably soundly decides if a Diophantine polynomial has integer roots, meaning:

$$p \text{ is } D_A\text{-decided} \iff p \text{ has no integer roots.}$$

We can of course also construct such a D_A more directly. Likewise there is a Turing decision machine that stably soundly decides the halting problem.

Definition 3.6. *Given a machine*

$$M : Dom \subset \mathbb{N} \times B \to \{\pm\}$$

and a Turing machine

$$T : Dom \subset \mathbb{N} \times B \to \{\pm\},$$

we say that T **stably computes** M*, or that* $\Theta^s_{M,T}$ *holds, if*

$$b \text{ is } M\text{-decided} \iff b \text{ is } T\text{-decided}.$$

4. Incompleteness for stably sound Turing machines

Let $\mathcal{D}$ denote the set of abstract machines of the form:

$$D : Dom \subset \mathcal{T} \times \mathbb{N} \to \{\pm\},$$

with subset Dom not fixed. And set

$$\mathcal{D}^t := \{T \in \mathcal{T}_{\mathcal{S}} | fog(T) \in \mathcal{D}\}.$$

In what follows, for $T \in \mathcal{T}$ when we write $T \in \mathcal{D}^t$ we mean that $T \in \text{image}\, i|_{\mathcal{D}^t}$, for $i : \mathcal{T}_{\mathcal{S}} \to \mathcal{T}$ the embedding discussed in Section 2.2. Likewise, if $T \in \text{image}\, i|_{\mathcal{D}^t}$ then by $T(T, m)$ we mean $i^{-1}(T)(T, m)$, so that in what follows the sentence "T is not T-decided" makes sense for such a T. Explicitly, for $T \in \text{image}\, i|_{\mathcal{D}^t}$, T is not T-decided, will mean that for every n with $i^{-1}(T)(T, n) = +$ there exists an $m > n$ such that $i^{-1}(T)(T, m) = -$.

Definition 4.1. *For a* $D \in \mathcal{D}$*, we say that* D *is* **stably sound on** $T \in \mathcal{T}$ *if*

$$(T \text{ is } D\text{-decided}) \implies (T \in \mathcal{D}^t) \wedge (T \text{ is not } T\text{-decided}).$$

We say that D *is* **stably sound** *if it is stably sound on all* T*. We say that* D **stably decides** $\mathcal{P}(T)$ *if whenever the statement:*

$$(T \in \mathcal{D}^t) \wedge (T \text{ is not } T\text{-decided})$$

is true, T *is* D*-decided. We say that* D **stably soundly decides** $\mathcal{P}(T)$ *if* D *is stably sound on* T *and stably decides* $\mathcal{P}(T)$*. We say that* D **stably soundly decides** $\mathcal{P}$ *if* D **stably soundly decides** $\mathcal{P}(T)$ *for all* $T \in \mathcal{T}$*.*

The interpretation of the above is that each $D \in \mathcal{D}$ is understood as a machine with the properties:

- For each T, n $D(T, n) = +$ if D decides at the moment n that $T \in \mathcal{D}^t$ and T is not T-decided .
- For each T, n $D(T, n) = -$ if D does not decide/assert at the moment n that $T \in \mathcal{D}^t$, or D does not decide that T is not T-decided.

Lemma 4.2. *If* D *is stably sound on* $T \in \mathcal{T}$ *then*

$$\neg\Theta^s_{D,T} \vee \neg(T \text{ is } D\text{-decided}).$$

Proof. If

$$T \text{ is } D\text{-decided}$$

then since D is stably sound on T, T is not T-decided, so of course $\neg\Theta^s_{D,T}$. □

Theorem 4.3. *There is no (stably) computable* $D \in \mathcal{D}$ *that stably soundly decides* $\mathcal{P}$*.*

Proof. Suppose that $D \in \mathcal{D}$ stably soundly decides $\mathcal{P}$ then by the above lemma we obtain:

$$\forall T \in \mathcal{D}^t : \Theta^s_{D,T} \implies \neg(T \text{ is } D\text{-decided}). \tag{4.4}$$

But it is immediate:

$$\forall T \in \mathcal{D}^t : \Theta^s_{D,T} \implies (\neg(T \text{ is } D\text{-decided}) \implies \neg(T \text{ is } T\text{-decided}))). \tag{4.5}$$

So combining (4.4), (4.5) above we obtain

$$\forall T \in \mathcal{D}^t : \Theta^s_{D,T} \implies \neg(T \text{ is } T\text{-decided}).$$

But D stably soundly decides $\mathcal{P}$ so we conclude:

$$\forall T \in \mathcal{D}^t : \Theta^s_{D,T} \implies (T \text{ is } D\text{-decided}).$$

But this is a contradiction to (4.4) unless

$$\forall T \in \mathcal{D}^t : \neg\Theta^s_{D,T},$$

which is what we wanted to prove. □

Definition 4.6. *For $D \in \mathcal{D}^t$, we say that $\mathcal{R}(D)$ holds if for any $T \in \mathcal{D}^t$ such T is not T-decided:*

$$\exists T' \in \mathcal{D}^t : (D \textit{ stably decides } \mathcal{P}(T')) \wedge (T \simeq_s T').$$

Theorem 4.7. *For $D \in \mathcal{D}$ the following cannot hold simultaneously: D is stably sound, D is (stably) computable and $\mathcal{R}_D$ holds.*

Proof. Suppose that D is stably computed by some $T \in \mathcal{D}^t$. If D is stably sound then by Lemma 4.2

$$\neg(T \text{ is } D\text{-decided}),$$

and so

$$\neg(T \text{ is } T\text{-decided}),$$

since T stably computes D. Consequently, if $\mathcal{R}(D)$ then for some $T' \in \mathcal{D}^t$ stably computing T, T' is D-decided. And so by Lemma 4.2 $\neg\Theta^s_{D,T'}$, which is a contradiction. □

Proof of Theorem 1.2. Suppose otherwise that we have such an M, so M is stably sound, computable and is such that

$$\forall \Sigma \in \mathcal{A} : \Sigma \text{ is true} \iff \Sigma \text{ is } M\text{-stable}.$$

It is well known that sentences in the language of Turing machines are logically equivalent to sentences in arithmetic. This means that given a sentence Σ^t in the language of Turing machines, e.g. "T halts on input n" there is a sentence $\Sigma \in \mathcal{A}$ so that Σ^t is true iff Σ is true. Indeed this kind of translation already appears in the original work of Turing [1].

Let $s(T) \in \mathcal{A}$ be the sentence logically equivalent to

$$(T \in \mathcal{D}^t) \wedge (T \text{ is not } T\text{-decided}).$$

Define a machine $\widetilde{D}_M \in \mathcal{D}$ by

$$\widetilde{D}_M(T, n) := D_M(s(T), n)$$

for D_M defined as in Section 3. Since M is stably sound, by assumption, $\widetilde{D}_M$ is stably sound by the second part of Lemma 3.5. By first part of Lemma 3.5 $\widetilde{D}_M \in \mathcal{D}$ is computable, and $\widetilde{D}_M$ stably decides $\mathcal{P}$ again by assumptions on M and Lemma 3.5. But this contradicts Theorem 4.3. □

5. Application: A human as a decision machine

Let S be a human subject in a controlled environment, in communication with an experimenter/operator E that as input passes to S elements of $\mathcal{I} = \mathcal{T} \times \mathbb{N}$. Here *controlled environment* means primarily that no information that is not explicitly controlled by E and that is usable by S passes to S while they are in this environment. This condition is only for simplicity, so long as we know in principle what "input" our S receives it doesn't matter what kind of environment they are in. For practical purposes S has in their environment a general purpose digital computer with arbitrarily, as necessary, expendable memory, (in other words a universal Turing machine).

We suppose that upon receiving any $I \in \mathcal{I}$, (for example as data in their computer) after possibly using their computer in some way, S instructs their computer to print after some indeterminate time an element

$$O \in \{\pm\}.$$

So S is meant to determine a decision machine:

$$D_S : \mathcal{T} \times \mathbb{N} \to \{\pm\}, \tag{5.1}$$

interpreted as in the previous section.

In what follows we will say S stably decides instead of saying D_S stably decides. If D_S is stably sound and computable then by Theorem 4.7 $\neg\mathcal{R}(D_S)$. And this in particular means that there exists a $T_S \in \mathcal{D}^t$ so that the statement

$$(T_S \in \mathcal{D}^t) \wedge (T_S \text{ is not } T_S\text{-decided})$$

is true but S will never stably decided it to be true. In fact S will never stably decide

$$(T' \in \mathcal{D}^t) \wedge (T' \text{ is not } T_S\text{-decided}),$$

for any $T' \in \mathcal{D}^t$ with $T' \simeq_s T_S$.

So if we define $\mathcal{H}(S) := s(T_S)$, with the latter defined as in the proof of Theorem 1.2, then we obtain our Theorem 1.3.

5.1. **Relationship with the Gödel and Penrose argument.** The most lucid analysis, known to me, of the Gödel and Penrose arguments for obstructions to computability of intelligence appears in Koellner [13], [14]. Our argument for Theorem 1.3 extends Gödel's idea [11, 310], although I am also inspired by the ideas of Penrose. One note is that our argument is entirely based on set theory, while Gödel's argument has meta-logical elements that require interpretation. Although, as Koellner explains [14], Gödel's argument can also be at least in some sense fully formalized.

This is not to say that there are no issues of interpretation in Theorem 1.3. One must interpret our definition of stable soundness as it applies to actual human beings. We of course have already partly addressed this. Scientists operate on the unshakeable faith that scientific progress converges on truth. And our interpretation above of this convergence as stable soundness is very simple and natural, at least under the previously explained assumption of weak idealization.

It is of course always the case that we must interpret mathematical theorems when applied to the real world. What one looks for is whether there is any meaningful physical obstruction to carrying out the necessary idealization in principle. In our specific case I see no such obstruction. Of course if the universe and humanity must eventually go extinct then our weakly idealized humans cannot even in principle exist. But to me this is not a meaningful obstruction. The potential mortality of the universe is very unlikely to have any causal relation with computability of intelligence. So we can imagine an eternal universe and a weakly idealized human, run the argument then translate to our universe.

So the only thing to reasonably wonder is whether for every S there must be such a stably undecidable arithmetic statement of the form $\mathcal{H}(S)$ above.

Acknowledgements. Dennis Sullivan, Bernardo Ameneyro Rodriguez, David Chalmers, and in particular Peter Koellner for helpful discussions on related topics.

References

[1] A.M. Turing, *On computable numbers, with an application to the entscheidungsproblem* , Proceedings of the London mathematical society, s2-42 (1937).
[2] ———, *Computing machines and intelligence*, Mind, 49 (1950), pp. 433–460.
[3] L. Blum, M. Shub, and S. Smale, *On a theory of computation and complexity over the real numbers: NP-completeness, recursive functions and universal machines.*, Bull. Am. Math. Soc., New Ser., 21 (1989), pp. 1–46.
[4] D. J. Chalmers, *Minds machines and mathematics*, Psyche, symposium, (1995).
[5] D. Deutsch, *Quantum theory, the Church-Turing principle and the universal quantum computer.*, Proc. R. Soc. Lond., Ser. A, 400 (1985), pp. 97–117.
[6] S. Feferman, *Are There Absolutely Unsolvable Problems? Gödel's Dichotomy*, Philosophia Mathematica, 14 (2006), pp. 134–152.
[7] C. Fields, D. Hoffman, C. Prakash, and M. Singh, *Conscious agent networks: Formal analysis and application to cognition*, Cognitive Systems Research, 47 (2017).
[8] P. Grindrod, *On human consciousness: A mathematical perspective*, Network Neuroscience, 2 (2018), pp. 23–40.
[9] S. Hameroff and R. Penrose, *Consciousness in the universe: A review of the 'orch or' theory*, Physics of Life Reviews, 11 (2014), pp. 39 – 78.
[10] J.R. Lucas, *Minds machines and Gödel*, Philosophy, 36 (1961).
[11] K. Gödel, *Collected Works III (ed. S. Feferman)*, New York: Oxford University Press, 1995.
[12] A. Kent, *Quanta and qualia*, Foundations of Physics, 48 (2018), pp. 1021–1037.

[13] P. Koellner, *On the Question of Whether the Mind Can Be Mechanized, I: From Gödel to Penrose*, Journal of Philosophy, 115 (2018), pp. 337–360.
[14] ———, *On the question of whether the mind can be mechanized, ii: Penrose's new argument*, Journal of Philosophy, 115 (2018), pp. 453–484.
[15] K. Kremnizer and A. Ranchin, *Integrated information-induced quantum collapse*, Foundations of Physics, 45 (2015), pp. 889–899.
[16] R. Penrose, *Beyond the shadow of a doubt*, Psyche, (1996).

University of Colima, Department of Sciences, CUICBAS
Email address: yasha.savelyev@gmail.com