INCOMPLETENESS FOR STABLY SOUND TURING MACHINES

YASHA SAVELYEV

Abstract. We first partly develop a mathematical notion of stable soundness intended to reflect the soundness property of (weakly idealized) 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 very non-trivial extension as 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 2.1.

Let $\mathcal{A}$ denote the set of (first order) sentences of arithmetic. Imagine a decision machine:

$$D : \mathcal{A} \times \mathbb{N} \to \{\pm\}, \tag{1.1}$$

with the following properties. $D(\Sigma, n) = +$ if at the moment n D decides that Σ is true. $D(\Sigma, m) = -$ if at the moment m D decides that Σ is false. If there is an n_0 with $D(\Sigma, n_0) = +$ s.t. there is no $m > n_0$ with $D(\Sigma, m) = -$ then we say that Σ ***is D-decided***, and that D ***decides*** or ***stably decides*** Σ to be true.

Definition 1.2. *We say that D is* **stably sound** *if every D-decided Σ is true. We say that D* **can decide** *the truth of any $\Sigma \in \mathcal{A}$ if every such true Σ is D-decided.*

So given the partial definitions above we may state our main theorem.

Theorem 1.3. *For a given decision machine M, of the type (1.1), the following cannot hold simultaneously: M is stably sound, M is computable, M can decide the truth of any arithmetic statement. Here* **computable** *has the mostly standard meaning of computability by a Turing machine, with specifics given in Section 2.1.*

If we replace stably sound by sound, then the above would just be a reformulation of Gödel's incompleteness theorem. (Replacing his consistency hypothesis by the stronger condition of soundness.) Moreover, if we specialize our argument to this case then in the statement of the theorem we may also specialize to pseudo-Diophantine arithmetic statements, specifically statements of the kind: the Turing machine $T : \mathbb{N} \to \mathbb{N}$ has no output. On the other hand there is a stably sound Turing machine M which can decide the truth of any such pseudo-Diophantine statement, cf. Example 4.3. So perhaps Turing machines stably soundly decide the truth of any arithmetic statement? The above theorem shows that this is not the case.

[1] 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.

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 algorithm 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 [12, 310], relating the question to existence of absolutely unsolvable Diophantine problems, see also Feferman [7], and Koellner [14], [15] 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 [15]. At the same time Gödel doubted that $\neg A$ is 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 4 specifically) then there is a Turing machine T whose stabilization T^s provably decides the halting problem, cf. Example 4.3, and so T^s is no longer a Turing machine. In that case, the above disjunction becomes meaningless because we introduced non-computability by passing to the idealization. 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

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

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 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 [11] and later again and more robustly Penrose [17] 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 [14], [15], argue that there are likely unresolvable meta-logical issues with the Penrose argument, even allowing for soundness. See also Penrose [17], 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.

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 1.3.

Theorem 1.4. *Either there are cognitively meaningful, absolutely non Turing computable processes in the human brain, or human beings are stably unsound, or for any (could be weakly idealized) 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. This theorem 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 [6], 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 1.5. *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* [10] *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:* [13], [16], [8], [9].

1.2. **Brief outline.** We first give some preliminaries on standard notions of Turing machines, and describe what we call abstractly encoded sets and abstract Turing machines. This likely already appeared in literature in a different guise, but these notions are elementary enough to make completely explicit here.

[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 4.3.

[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 $\mathbb{Z}$.

Next we isolate a certain class of (abstract) Turing machines that we name diagonalization machines. They print strings with a certain property C. As the name suggests their behavior is related to the Cantor diagonalization argument. Then we explicitly construct a certain special "Cantor string" $\mathcal{G}$ universal for this whole class. This string $\mathcal{G}$ has property C but cannot be printed by a Turing diagonalization machine. Crucially, this is then extended to stable diagonalization machines that print property C strings only stably. (It is not exactly property C but a certain stabilized version of this property.) This $\mathcal{G}$ is then used for our main results. Strictly speaking we can obtain our main results more directly, but most of the setup needed for the construction of $\mathcal{G}$ would still be necessary, and using $\mathcal{G}$ makes the argument more conceptual. In addition it may be of independent interest.

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**. *In this case we denote by s_f the final configuration, so that the sequence is eventually constant with terms s_f. We define the* **length** *of a computation sequence to be the first occurrence of $n > 0$ s.t. $s_n = s_f$. For a given Turing computation $*M(\Sigma)$, we will write*

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

if $*M(\Sigma)$ halts and x is the 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 T determines a partial function that is defined on all $\Sigma \in Strings$ s.t. $*T(\Sigma)$ halts, by $\Sigma \mapsto T(\Sigma)$.

Definition 2.4. *Given a partial function $f : Strings \to Strings$, we say that a Turing machine T* **computes** *f if*

$$T = f$$

as partial functions.

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

$$\Sigma \notin I \subset Strings,$$

where I is some set of all valid, that is T-***permissible*** input strings. We do not ask that for $\Sigma \in I$ $*T(\Sigma)$ halts. If $*T(\Sigma)$ does halt then we will say that Σ is T-***acceptable***.

Definition 2.5. *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

$$T : I \to O,$$

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

2.1. **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 if the input set is $Strings^2$, we may encode it 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 use often, 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 $A \in Set$, and morphisms $(m, \widetilde{m})$ commutative diagrams:

$$\begin{array}{ccc} A & \xrightarrow{\quad\widetilde{m}\quad} & B \\ \downarrow{\scriptstyle e_A} & & \downarrow{\scriptstyle e_B} \\ e_A(A) \subset Strings & \xrightarrow{\quad m\quad} & e_B(B) \subset Strings. \end{array}$$

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.

- For $A \in \mathcal{S}$ $e_A(A)$ is computable (recursive).
- For $A, B \in \mathcal{S}$ $A \sqcup B$ is in $\mathcal{S}$. Moreover, corresponding to the inclusion map $\widetilde{i}_A : A \to A \sqcup B$ we have a morphism $(i_A, \widetilde{i}_A)$ in $\mathcal{S}$:

$$\begin{array}{ccc} A & \xrightarrow{\widetilde{i}_A} & A \sqcup B \\ \downarrow & & \downarrow \\ e_A(A) & \xrightarrow{i_A} & e_{A\sqcup B}(A \sqcup B), \end{array}$$

  likewise with $\widetilde{i}_B : B \to A \sqcup B$. We ask that the map

$$e_A(A) \sqcup e_B(B) \to e_{A\sqcup B}(A \sqcup B)$$

  induced by $(i_A, \widetilde{i}_A)$, $(i_B, \widetilde{i}_B)$ is bijective and $e_A(A)$, $e_B(B)$ are computable, i.e. recursive sets.
- If $A, B \in \mathcal{S}$ then $A \times B \in \mathcal{S}$ and the projection maps $A \times B \to A$, $A \times B \to B$ complete to morphisms of $\mathcal{S}$ similar to the diagram above.
- 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. We may of course also write $e_A(A) \subset \mathcal{U}$.
- 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 $e_A(A)$.

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 $\mathbb{Z}, \mathbb{N}, \{\infty\}, \{\hbar\}, \{\epsilon\}$ $\{\pm\}, \mathcal{T}$, with $\{\infty\}, \{\hbar\}, \{\epsilon\}$ abstract singletons, $\{\pm\}$ a set with two elements $+, -$. The encodings of $\mathbb{N}$, $\mathbb{Z}$ and $\mathcal{T}$ should be suitably natural so that for example there is a computable bijective map $S : e(\mathbb{Z}) \to e(\mathbb{Z})$ which corresponds under e to the self-map $n \mapsto n + 1$ of $\mathbb{Z}$. This is not part of the axioms but we need this to construct specific abstract Turing machines, whose definition follows.

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. *For $A, B \in \mathcal{S}$ an* **abstract Turing machine**

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

is a partial map $f : A \to B$, and a Turing machine

$$T_e : e_A(D) \subset e_A(A) \to e_B(B)$$

such that T_e computes f_e, with $e_A(D)$ the set of T_e-permissible strings. The set D is called the set of **T-permissible strings**. *We often omit D from notation, especially when $D = A$. In practice the partial map f will 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. We may by abuse of notation identify $\mathcal{T}_{\mathcal{S}}$ with its image $i(\mathcal{T}_{\mathcal{S}}) \subset \mathcal{T}$.

For writing purposes we condense the above as follows.

Definition 2.7. *A* **machine** [6] *will be a synonym for a partial map $M : D \subset A \to B$, with A, B abstractly encoded sets. D may be omitted from notation.*

[6] 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.

$\mathcal{M}$ 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 $fog(T) = M$.*

Composition. Given (abstract) Turing machines

$$M_1 : I \to O, M_2 : J \to P,$$

we may naturally ***compose*** them to get a Turing machine $M_2 \circ M_1 : C \to P$, for $C = M_1^{-1}(O \cap J)$, ($O \cap J$ is understood as intersection of subsets of $Strings$). C can be empty in which case this is a Turing machine which rejects all input. Let us not elaborate further.

2.2. **Join of Turing machines.** Our Turing machine of Definition 2.1 is a multi-tape enhancement of a more basic notion of a Turing machine with a single tape, but we need to iterate this further.

We replace a single tape by tapes $T^1, \dots, T^n$ in parallel, which we denote by $(T^1 \dots T^n)$ and call this n-tape. The head H on the n-tape has components H^i pointing on the corresponding tape T^i. When moving a head we move all of its components separately. A string of symbols on $(T^1 \dots T^n)$ is an n-string, formally just an element $\Sigma \in Strings^n$, with i'th component of Σ specifying a string of symbols on T^i. The blank symbol b is the symbol $(b^1, \dots, b^n)$ with b^i blank symbols of T^i.

Given Turing machines M^1, M^2 we can construct what we call a ***join*** $M^1 \star M^2$, which is roughly a Turing machine where we alternate the operations of M^1, M^2. In what follows symbols with superscript $1, 2$ denote the corresponding objects of M^1, respectively M^2, cf. Definition 2.1.

$M^1 \star M^2$ has three 2-tapes:

$$(T_i^1 T_i^2), (T_c^1 T_c^2), (T_o^1 T_o^2),$$

three heads H_i, H_c, H_o which have component heads H_i^j, H_c^j, H_o^j, $j = 1, 2$. It has machine states:

$$Q_{M^1 \star M^2} = Q^1 \times Q^2 \times (\mathbb{Z}_2 = \{0, 1\}),$$

with initial state $(q_0^1, q_0^2, 0)$ and final states:

$$F_{M^1 \star M^2} = F^1 \times Q^2 \times \{1\} \sqcup Q^1 \times F^2 \times \{0\}.$$

Clearly we have a natural splitting

$$\mathcal{C}(M^1 \star M^2) = \mathcal{C}(M^1) \times \mathcal{C}(M_2) \times \mathbb{Z}_2.$$

In terms of this splitting we define the transition function

$$\delta^{M^1 \star M^2} : \mathcal{C}(M^1 \star M^2) \to \mathcal{C}(M^1 \star M^2),$$

for our Turing machine $M^1 \star M^2$ by:

$$\delta^{M^1 \star M^2}(s^1, s^2, 0) = (\delta^{M^1}(s_1), s^2, 1)),$$
$$\delta^{M^1 \star M^2}(s^1, s^2, 1) = (s_1, \delta^{M^2}(s^2), 0)).$$

Or, concretely this means the following. Given machine state $q = (q^1, q^2, 0)$ and the symbols

$$(\sigma_i^1 \sigma_i^2), (\sigma_c^1 \sigma_c^2), (\sigma_o^1 \sigma_o^2)$$

to which the heads H_i, H_c, H_o are currently pointing, we first check instructions in I^1 for q^1, $\sigma_i^1, \sigma_c^1, \sigma_o^1$, and given those instructions as step 1 execute:

(1) Replace symbols σ_c^1, σ_o^1 to which the head components H_c^1, H_o^1 point, or leave them unchanged, while leaving unchanged the symbols to which H_c^2, H_o^2 point.
(2) Move each head component H_i^1, H_c^1, H_o^1 left, right, or leave it in place, (independently). (The second components of the heads are unchanged.)

(3) Change the first component of q to another machine state in Q^1 or keep it, based on the instruction in I^1. Leave the second component of q unchanged. The third component of q is changed to 1.

Then likewise given machine state $q = (q^1, q^2, 1)$, we check instructions in I^2 for q^2, $\sigma_i^2, \sigma_c^2, \sigma_o^2$ and given those instructions as step 2 execute:

(1) Replace symbols σ_c^2, σ_o^2 to which the head components H_c^2, H_o^2 point, or leave them unchanged, while leaving unchanged the symbols to which H_c^1, H_o^1 point.
(2) Move each head component H_i^2, H_c^2, H_o^2 left, right, or leave it in place.
(3) Change the second component of q to another or keep it, based on instruction in I^2. Leave the first component unchanged, and change the third component of q to 0.

2.2.1. *Input.* The input for $M^1 \star M^2$ is a 2-string or in other words pair (Σ_1, Σ_2), with Σ_1 an input string for M^1, and Σ_2 an input string for M^2.

2.2.2. *Output.* The output for

$$*M^1 \star M^2(\Sigma_1, \Sigma_2)$$

is defined as follows. If this computation halts then the 2-tape $(T_o^1 T_o^2)$ contains a 2-string, bounded by b symbols, with T_o^1 component Σ_o^1 and T_o^2 component Σ_o^2. Then the output $M^1 \star M^2(\Sigma_1, \Sigma_2)$ is defined to be Σ_o^1 if the final state is of the form $(q_f, q, 1)$ for q_f final, or Σ_o^2 if the final state is of the form $(q, q_f, 0)$, for q_f likewise final.

So we can understand $M^1 \star M^2$ as an abstract Turing machine:

$$M^1 \star M^2 : \mathcal{U} \times \mathcal{U} \to \mathcal{U}.$$

2.3. **Universal Turing machines.** It will be convenient to refer to the (in our language abstract) universal Turing machine

$$U : \mathcal{T} \times \mathcal{U} \to \mathcal{U}.$$

This universal Turing machine already appears in Turing's [1]. As a partial map it is defined by

$$U(T, \Sigma) = T(\Sigma),$$

if we identify $\mathcal{T}$ with the set of abstract Turing machines $\mathcal{U} \to \mathcal{U}$, tautologically.

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

$$T : I \to O,$$

with the full data of the underlying Turing machine being implicitly specified, in a way that should be clear from context. When we intend to suppress dependence of a variable V on some parameter p we often write $V = V(p)$, this equality is then an equality of notation not of mathematical objects.

3. Diagonalization machines and Cantor strings

This section can be understood to be a warm up, as we will not yet work with stable soundness. But we will still need most of the elements of this section.

3.1. **Diagonalization machines.** There is a well known connection between Turing machines and formal systems, see for instance [7]. So Gödel statements can already be interpreted in Turing machine language as certain strings. It will help to partly develop a certain flexible analogue of such strings. This will then be extended to stably sound setting. To this end we need set up a certain class of Turing machines that we call diagonalization machines.

We denote by $\mathcal{T}_{\mathbb{Z}} \subset \mathcal{T}_{\mathcal{S}}$ the subset corresponding to abstract Turing machines of the type:

$$X : (S_X \times \mathbb{N} \subset \mathcal{U} \times \mathbb{N}) \to \mathbb{Z}.$$

Let $\mathcal{O} \subset \mathcal{T}_{\mathbb{Z}} \times \mathcal{U}$ consist of (X, Σ) with $\Sigma \in S_X$, for S_X as above. And set

$$\mathcal{O}' := \mathcal{O} \times \mathbb{N} \subset \mathcal{T}_{\mathbb{Z}} \times \mathcal{U} \times \mathbb{N}.$$

Recall also that we identify $\mathcal{T}_{\mathcal{S}}$ as a subset of $\mathcal{T}$ via the map i. So while the sets $\mathcal{O}, \mathcal{O}'$ are not abstractly encoded they are identified as subsets of the abstractly encoded sets $\mathcal{T} \times \mathcal{U}$, $\mathcal{T} \times \mathcal{U} \times \mathbb{N}$ respectively, by encoding of e $\mathcal{O}, \mathcal{O}'$ we will mean restriction of the encodings of these ambient sets.

Let

$$D_1 : \mathbb{Z} \sqcup \{\infty\} \to \mathbb{Z},$$

be a fixed Turing machine which satisfies

$$D_1(x) = x + 1 \text{ if } x \in \mathbb{Z} \subset \mathbb{Z} \sqcup \{\infty\} \tag{3.1}$$

$$D_1(\infty) = 1. \tag{3.2}$$

Here $\{\infty\}$ is the one point set containing the element ∞. In what follows we sometimes understand D_1 as an element of $\mathcal{T}_{\mathbb{Z}}$, denoting the Turing machine:

$$(x, m) \mapsto D_1(x), \tag{3.3}$$

for all $(x, m) \in (\mathbb{Z} \sqcup \{\infty\}) \times \mathbb{N}$.

We need one more Turing machine.

Definition 3.4. *We say that a (abstract) Turing machine*

$$R : D \subset \mathcal{T} \times \mathcal{U} \times \mathbb{N} \to \mathbb{Z} \sqcup \{\infty\},$$

has **property** G *if the following is satisfied:*

- *R is defined on all of $\mathcal{O}'$, in particular $D \supset \mathcal{O}'$.*
- *$R(X, \Sigma, m) \neq \infty \implies R(X, \Sigma, m) = X(\Sigma, m)$, for $(\Sigma, m) \in S_X \times \mathbb{N}$, and $X \in \mathcal{T}_{\mathbb{Z}}$.*
- *$\forall m : R(D_1, \infty, m) \neq \infty$, and so $\forall m : R(D_1, \infty, m) = 1$, by the previous property.*

Lemma 3.5. *There is a Turing machine R satisfying property G.*

Proof. Let W_n be some Turing machine $W_n : \{\epsilon\} \to \{\infty\}$, for $\epsilon \in \mathcal{U}$ the empty string. So as a function it is not very interesting since the input and output sets are singletons. We ask that the length of $*W_n(\epsilon)$ is $n > 0$, (cf. Preliminaries). Let

$$R_n : D \subset \mathcal{T} \times (\mathcal{U} \times \mathbb{N}) \to \mathcal{U}$$

be the Turing machine specified as:

$$R_n(Z) := (W_n \star U)(\epsilon, e(Z)),$$

for $e : \mathcal{U} \times \mathbb{N} \to \mathit{Strings} = \mathcal{U}$ the encoding map, for $\star$ the join operation described in Section 2.2, for U the universal Turing machine, as described. Clearly R_n always halts, although it may halt with machine state q_{reject}. Moreover by construction every

$$Z \in \mathcal{O}'$$

is R_n-permitted. Additionally, for such a $Z = (X, (\Sigma, m)) \in \mathcal{O}'$

$$R_n(X, (\Sigma, m)) \neq \infty \implies R_n(X, (\Sigma, m)) = X(\Sigma, m),$$

in particular R_n is defined on such a Z.

As a map $\mathbb{Z} \sqcup \{\infty\} \to \mathbb{Z}$ D_1 is completely determined but it could have various implementations as a Turing machine, so that the length l_m of $*D_1(\infty, m)$ depends on this implementation. Clearly we may assume that $\forall m : l = l_m$ for some l, by definition of D_1 as an element of $\mathcal{T}_{\mathbb{Z}}$, as in (3.3). We then ask that $n_0 > l$ is fixed. Then by construction we get:

$$\forall m : R_{n_0}(D_1, \infty, m) = D_1(\infty, m) = 1.$$

So set $R := R_{n_0}$, and this gives the desired Turing machine.

Note that the domain D of R-permissible strings is not explicitly determined by our construction, as we cannot tell without additional information when a general Z is rejected by R. We can only say that $D \supset \mathcal{O}'$. □

Define $\mathcal{M}_0$ to be the set of machines whose input set is $\mathcal{I} = \mathcal{T} \times \mathbb{N}$ and whose output set is $\mathcal{U}$. That is

$$\mathcal{M}_0 := \{M \in \mathcal{M} | M : \mathcal{T} \times \mathbb{N} \to \mathcal{U}\}.$$

We set

$$\mathcal{T}_0 := \{T \in \mathcal{T}_{\mathcal{S}} | fog(T) \in \mathcal{M}_0\},$$

and we set $\mathcal{I}_0 := \mathcal{T}_0 \times \mathbb{N}$. Given $M \in \mathcal{M}_0$ and $M' \in \mathcal{T}_0$ let $\Theta_{M,M'}$ be the statement:

$$M \text{ is computed by } M'. \tag{3.6}$$

For each $M \in \mathcal{M}_0$, we define a machine:

$$\widetilde{M} : \mathcal{I} \to \mathcal{U} \times \mathbb{N}$$

$$\widetilde{M}(B, m) = (M(B, m), m). \tag{3.7}$$

This is naturally a Turing machine when M is a Turing machine. In what follows, for $T \in \mathcal{T}_0$, when we write $T(T, m)$ we mean $T(i(T), m))$ where $i : \mathcal{T}_{\mathcal{S}} \to \mathcal{T}$ is as in Section 2.1.

Definition 3.8. *For $M \in \mathcal{M}_0$, $T \in \mathcal{T}_0$, $O \in Strings$ is said to have* **property** $C = C(M, T)$ *if:*

$$\Theta_{M,T} \implies \forall m : (T(T, m) \notin e(\mathcal{O})) \vee (T(T, m) \in e(\mathcal{O}), O \in e(\mathcal{O}) \text{ and } X(\Sigma, m) = D_1 \circ R \circ \widetilde{T}(T, m)),$$

where $(X, \Sigma) \in \mathcal{O}$, and $e(X, \Sigma) = O$, and where $\widetilde{T}$ is determined by T as in (3.7). Here e refers to relevant encoding map.

Notation 1. It is cumbersome to always write various encoding maps, as well identifications via the map i, so from now on, in the rest of the paper, both are omitted, as they should be clear from context.

At a glance, this is a somewhat complicated property, but essentially it just says that if $\Theta_{M,T}$ then for all m "$O \neq T(T, m)$" unless either $T(T, m)$ is undefined, or the output does not have the right (data) type, or $R(O, m) = \infty$. Thus the string O with property $C(M, T)$ is "diagonal" in a certain sense, where by "diagonal" we mean that something analogous to Cantor's diagonalization is happening, but we will not elaborate.

Remark 3.9. *The fact that data types get intricated is perhaps not surprising. On one hand there is a well known correspondence, the Curry-Howard correspondence [5], between proof theory in logic and type theory in computer science, and on the other hand we are doing something at least loosely related to Gödel incompleteness, but in the language of Turing machines.*

Definition 3.10. *We say that $M \in \mathcal{M}_0$ is* **C-sound**, *or is a* **diagonalization machine**, *if for each $(T, m) \in \mathcal{I}_0$, with $M(T, m) = O$ defined, O has property $C(M, T)$. We say that M is C-sound on T if the list $\{M(T, m)\}_m$ has only elements with property $C(M, T)$.*

Define a C-sound $T \in \mathcal{T}_0$ analogously.

Definition 3.11. *If M as above is C-sound we will say that sound(M) holds. If M is C-sound on T we say that sound(M, T) holds.*

Example 3.12. A trivially C-sound machine M is one for which

$$M(T, m) = (D_1 \circ R \circ \widetilde{T}, T)$$

for every $(T, m) \in \mathcal{I}$. As $(D_1 \circ R \circ \widetilde{T}, T)$ automatically has property $C(M, T)$ for each $T \in \mathcal{T}_0$. In general, for any $M \in \mathcal{M}_0$, $T \in \mathcal{T}_0$ the list of all strings O with property $C(M, T)$ is always infinite, as by this example there is at least one such string $(D_1 \circ R \circ \widetilde{T}, T)$, which can then be modified to produce infinitely many such strings.

Theorem 3.13. *Given any $M \in \mathcal{M}_0$, if $sound(M, M') \wedge \Theta_{M,M'}$ for some $M' \in \mathcal{T}_0$ then*

$$\forall m : M(M', m) \neq \mathcal{G},$$

where $\mathcal{G} := (D_1, \infty)$. On the other hand:

$$\forall T \in \mathcal{T}_0 : sound(T, T) \implies \mathcal{G} \text{ has property } C(T, T).$$

In particular if $sound(M)$ then $\mathcal{G}$ has property $C(M, T)$ for all $T \in \mathcal{T}_0$.

In the second half of the above theorem we may implicitly treat an element $T \in \mathcal{T}_0$ as an element of $\mathcal{M}_0$ via the map fog. So given any C-sound $M \in \mathcal{M}_0$ there is a certain string $\mathcal{G}$ with property $C(M, T)$ for all $T \in \mathcal{T}_0$, such that for each $M' \in \mathcal{T}_0$ if $\Theta_{M,M'}$ then

$$\mathcal{G} \neq M(M', m),$$

for all m. This $\mathcal{G}$, named Cantor string, is what we are going to extend to stably sound setting in the next sections. What makes $\mathcal{G}$ particularly suitable for our application is that it is independent of the particulars of M, all that is needed is $\mathcal{M} \in \mathcal{M}_0$ and is C-sound. So $\mathcal{G}$ is in a sense universal.

Proof. Suppose not and let M_0' be such that $\Theta_{M,M_0'} \wedge sound(M, M_0')$ and such that

$$M(M_0', m_0) = \mathcal{G} \text{ for some } m_0,$$

so that $\mathcal{G}$ has property $C(M, M_0')$. Set $I_0 := (M_0', m_0)$ then we have that:

$$\begin{aligned}
&1 = D_1(\infty, m_0).\\
&D_1(\infty, m_0) = D_1 \circ R \circ \widetilde{M'}(I_0), \text{ by } \mathcal{G} \text{ having property } C(M, M')\\
&\text{and by } M'(I_0) = \mathcal{G} \in \mathcal{O} \text{ since } \Theta_{M,M'}.\\
&D_1 \circ R \circ \widetilde{M'}(I_0) = D_1 \circ R(D_1, \infty, m_0) \quad \text{by } M'(I_0) = \mathcal{G}.\\
&D_1 \circ R(D_1, \infty, m_0) = 2 \quad \text{by property } G \text{ of } R \text{ and by (3.1)}.\\
&1 = 2.
\end{aligned}$$

So we obtain a contradiction.

We now verify the second part of the theorem. We show that:

$$\forall m, \forall T \in \mathcal{T}_0 : \Big(sound(T, T) \wedge (T(T, m) \in \mathcal{O}) \implies R(\widetilde{T}(T, m)) = \infty\Big). \tag{3.14}$$

Suppose otherwise that for some m_0, T_0 and $I_0 := (T_0, m_0)$ we have:

$$sound(T_0, T_0) \wedge (T_0(I_0) \in \mathcal{O}) \wedge (R(\widetilde{T_0}(I_0)) \neq \infty).$$

So we have:

$$T_0(I_0) = (X, \Sigma) \in \mathcal{O}, \tag{3.15}$$

for some (X, Σ) having property $C(T_0, T_0)$, by $sound(T_0, T_0)$. And so, since R is defined on all of $\mathcal{O}'$:

$$R(\widetilde{T_0}(I_0)) = R(X, \Sigma, m_0) = X(\Sigma, m_0) = x \in \mathbb{Z}, \text{ for some } x,$$

by Property G of R and by $R(\widetilde{T_0}(I_0)) \neq \infty$.

Then we get:

$$x = X(\Sigma, m_0) = D_1 \circ R \circ \widetilde{T_0}(I_0) = D_1(x) = x + 1$$

by (X, Σ) having property $C(T_0, T_0)$, and by (3.15). So we get a contradiction and (3.14) follows. Our conclusion readily follows.

The last part of the theorem follows by logic, for we have:

$$\forall T \in \mathcal{T}_0 : sound(T, T) \implies \mathcal{G} \text{ has property } C(T, T).$$

Also

$$\forall T : \Theta_{M,T} \wedge sound(M) \implies sound(T, T),$$

therefore

$$\forall T : sound(M) \implies (\Theta_{M,T} \implies \mathcal{G} \text{ has property } C(T, T)).$$

Which is the same as:

$$\forall T : sound(M) \implies \mathcal{G} \text{ has property } C(M,T),$$

by the definition of property $C(M,T)$.

□

4. Stable Cantor strings

Definition 4.1. *Given a machine:*

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

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

Definition 4.2. *Given a machine*

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

we define

$$M^s : \mathbb{N} \to \mathcal{U}$$

to be the machine enumerating, in order, all the M-stable Σ. 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 4.3. 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 whose stabilization enumerates Turing machines (with input the empty string ϵ) which do not halt. These sets are well known to be not computably enumerable, [1].

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 $\{N(0), \dots, N(n)\}$ are solutions of p. Whenever no add $(p, +)$ to L, whenever yes add $(p, -)$.
- Set $n := n + 1$ go to Start and continue.

This will define a partial 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 which is L after the m'th step of the algorithm. It's stabilization A^s enumerates Diophantine polynomials which have no integer roots.

The crucial point of our Cantor string is that it will still function in the context of this "stability". Let $\mathcal{M}^{\pm}$ denote the set of machines

$$M : \mathcal{I} = \mathcal{T} \times \mathbb{N} \to \mathcal{U} \times \{\pm\}.$$

We set

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

Definition 4.4. *Given a machine*

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

*we say that $b \in B$ is (M, a)-***stable** *if there exists an $m \in \mathbb{N}$ s.t. $M(a, m) = (b, +)$ and there is no $k > m$ s.t. $M(a, k) = (b, -)$.*

When $T \in \mathcal{T}^{\pm}$ is a Turing machine and $fog(T) = M$ instead of writing (M,T)-stable we may just write T-stable. Let

$$pr : \mathcal{U} \times \{\pm\} \to \mathcal{U},$$

be the natural projection. For each $M \in \mathcal{M}^{\pm}$ we define a machine:

$$\widetilde{M} : \mathcal{I} \to \mathcal{U} \times \mathbb{N},$$

$$\widetilde{M}(T,m) = (pr \circ M(T,m), m), \tag{4.5}$$

which is naturally a Turing machine when M is a Turing machine.

Definition 4.6. *Given a machine*

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

and a Turing machine

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

we say that T **stably computes** *M, or $\Theta^s_{M,T}$, if $M^s = T^s$ for T^s the stabilization of the machine $fog(T)$. Extend this definition naturally to machines of the form*

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

So that a Turing machine $T : A \times \mathbb{N} \to B \times \{\pm\}$ stably computes M if for each $a \in A$, $T_a = T|_{\{a\times\mathbb{N}\}}$ stably computes M_a.

In what follows $\mathcal{O} \subset \mathcal{T}_{\mathbb{Z}} \times \mathcal{U}$ is as before. Also recall that we are omitting to write encoding maps, as well as the identification of $\mathcal{T}_{\mathcal{S}}$ as a subset of $\mathcal{T}$.

Definition 4.7. *For $M \in \mathcal{M}^{\pm}$, $T \in \mathcal{T}^{\pm}$, $O \in Strings$ is said to have property $sC = sC(M,T)$ if:*

$$\Theta^s_{M,T} \implies \forall m : (pr \circ T(T,m) \notin \mathcal{O}) \vee (pr \circ T(T,m) \text{ is not } T\text{-stable})$$
$$\vee\, (pr \circ T(T,m) \in \mathcal{O}, O \in \mathcal{O} \text{ and } X(\Sigma,m) = D_1 \circ R \circ \widetilde{T}(T,m), \text{ where } (X,\Sigma) = O)),$$

for $\widetilde{T}$ determined by T as in (4.5).

Definition 4.8. *We say that $M \in \mathcal{M}^{\pm}$ is* **stably C-sound** *on $T \in \mathcal{T}$, and we write that $s - sound(M,T)$ holds, if every (M,T)-stable O has property $sC(M,T)$. We say that M is* **stably C-sound** *if it is stably C-sound on all T, and in this case we write that $s - sound(M)$ holds.*

Example 4.9. As before an example of a trivially stably C-sound machine M is one for which

$$M(T,m) = (D_1 \circ R \circ \widetilde{T}, T, +)$$

for every $(T,m) \in \mathcal{I}$.

Theorem 4.10. *For all $M \in \mathcal{M}^{\pm}$:*

$$(\exists M' \in \mathcal{T}^{\pm} : s - sound(M,M') \wedge \Theta^s_{M,M'}) \implies ((O \text{ is } (M,M')\text{-stable}) \implies O \neq \mathcal{G})$$

where

$$\mathcal{G} := (D_1, \infty) \in \mathcal{O}.$$

On the other hand,

$$\forall T \in \mathcal{T}^{\pm} : s - sound(T,T) \implies \mathcal{G} \text{ has property } sC(T,T). \tag{4.11}$$

Proof. This is mostly analogous to the proof of Theorem 3.13. Suppose not, let M be fixed and let M'_0 be such that $s - sound(M,M'_0) \wedge \Theta^s_{M,M'_0}$ and such that for some m_0:

$$M(M'_0, m_0) = (\mathcal{G}, +) \text{ so that } \nexists n > m_0 : M(M'_0, n) = (\mathcal{G}, -).$$

In particular $\mathcal{G}$ has property $sC(M,M'_0)$ by $s - sound(M,M'_0)$.

By Θ^s_{M,M'_0} there exists $m'_0 > 0$ such that $M'_0(M'_0, m'_0) = (\mathcal{G}, +)$. If we set $I_0 := (M'_0, m'_0)$, then by $\mathcal{G}$ having property $sC(M,M'_0)$, by $\mathcal{G} \in \mathcal{O}$ and by $\mathcal{G}$ being M'_0-stable as $\mathcal{G}$ is (M,M'_0)-stable:

$$D_1(\infty, m_0) = D_1 \circ R \circ \widetilde{M'_0}(I_0). \tag{4.12}$$

On the other hand:

$$D_1 \circ R \circ \widetilde{M}'_0(I_0) = D_1 \circ R(D_1, \infty, m_0) \quad \text{by } M'_0(I_0) = (\mathcal{G}, +), \tag{4.13}$$

$$D_1 \circ R(D_1, \infty, m_0) = 2 \quad \text{by property } G \text{ of } R \text{ and by (3.1)}, \tag{4.14}$$

$$D_1(\infty, m_0) = 1, \tag{4.15}$$

$$1 = 2, \text{by (4.12) and by (4.14)}. \tag{4.16}$$

So we obtain a contradiction.

We now verify the second part of the theorem. Given any $T \in \mathcal{T}^{\pm}$, for any $m \in \mathbb{N}$, setting $I := (T, m)$ we show that:

$$s - sound(T, T) \wedge (pr \circ T(I) \in \mathcal{O}) \wedge (pr \circ T(I) \text{ is } T\text{-stable}) \implies R(\widetilde{T}(I)) = \infty. \tag{4.17}$$

Suppose otherwise that for some T_0, m_0 and $I_0 := (T_0, m_0)$ we have:

$$s - sound(T_0, T_0) \wedge (pr \circ T_0(I_0) \in \mathcal{O}) \wedge (pr \circ T_0(I_0) \text{ is } T_0\text{-stable}) \wedge (R(\widetilde{T}_0(I_0)) \neq \infty).$$

Then by the above condition we get:

$$T_0(I_0) = (O, +), \text{ or } T_0(I_0) = (O, -), \tag{4.18}$$

for some $O = (X, \Sigma) \in \mathcal{O}$ that is (T_0, T_0)-stable and with property $sC(T_0, T_0)$, by $s - sound(T_0, T_0)$. We can of course guarantee that there is some m'_0 with $T_0(T_0, m'_0) = (O, +)$ but we arranged the details so that this is not necessary.

Since R is defined on all of $\mathcal{O}'$ we get:

$$R(\widetilde{T}_0(I_0)) = R(O, m_0) = X(\Sigma, m_0) = x \in \mathbb{Z}, \text{ for some } x,$$

by Property G of R and by $R(\widetilde{T}_0(I_0)) \neq \infty$. Then we have:

$$x = X(\Sigma, m_0) = D_1 \circ R \circ \widetilde{T}_0(I_0) = D_1(x) = x + 1,$$

by (X, Σ) having property $sC(T, T)$, and by (4.18). So we get a contradiction and (4.17) follows. Our conclusion readily follows. □

5. Incompleteness for stably sound Turing machines

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

$$D : \mathcal{T} \times \mathbb{N} \to \mathcal{U} \times \{\pm\} \bigsqcup \{\hbar\} \times \{\pm\} \bigsqcup \{\epsilon\},$$

with each D a total map. We may interpret elements $D \in \mathcal{D}$ as decision machines with the following properties.

- For each T, n $D(T, n) = (O, +)$, with $O \in \mathcal{U}$, if D asserts at the moment n that $T \in \mathcal{T}^{\pm}$ and O is (T, T)-stable.
- For each T, n $D(T, n) = (\hbar, +)$ if D asserts at the moment n that either $T \notin \mathcal{T}^{\pm}$ or no O is (T, T)-stable.
- For each T, n $D(T, n) = (\hbar, -)$ if D no longer asserts at the moment n that $T \notin \mathcal{T}^{\pm}$ and D no longer asserts at the moment n that no O is (T, T)-stable.
- For each T, n $D(T, n) = (O, -)$, with $O \in \mathcal{U}$, if either D no longer asserts at the moment n that $T \in \mathcal{T}^{\pm}$ or D no longer asserts at the moment n that O is (T, T)-stable.
- For each T, n $D(T, n) = \epsilon$ if D at the moment n does not assert anything new.

Definition 5.1. *We say that D is* **stably sound** *if for each T:*

$$\hbar \text{ is } (D, T)\text{-stable} \implies T \notin \mathcal{T}^{\pm} \text{ or no } O \text{ is } (T, T)\text{-stable}.$$

$$O \in \mathcal{U} \text{ is } (D, T)\text{-stable} \implies T \in \mathcal{T}^{\pm} \text{ and } O \text{ is } (T, T)\text{-stable}.$$

We say that D **stably decides** *$\mathcal{P}(T)$ if either $\hbar$ or some O as above is (D, T)-stable. We say that D* **stably soundly decides** *$\mathcal{P}(T)$ if D is stably sound on T and stably decides $\mathcal{P}(T)$.*

For each $D \in \mathcal{D}$ there is an associated element $M_D \in \mathcal{M}^{\pm}$ that is a machine

$$M_D : \mathcal{T} \times \mathbb{N} \to \mathcal{U} \times \{\pm\},$$

defined via the following meta-algorithm, which is a computational algorithm if D is a Turing machine.

- Let L be initialized as an empty set, which as before as an ordered list of instructions. Also initialize $n := 0$, and let $T \in \mathcal{T}$ be given.
- Start. If
$$D(T, n) = (O, +) \in \mathcal{U} \times \{\pm\} \text{ and } O \neq \mathcal{G}$$
then add $(\mathcal{G}, +)$ to L. If $O = \mathcal{G}$ then add $(O, -)$. If
$$D(T, n) = (\hbar, +)$$
then add $(\mathcal{G}, +)$ to L. If
$$D(T, n) = (\hbar, -)$$
then add $(\mathcal{G}, -)$ to L. Finally, if $D(T, n) = \epsilon$ then do nothing.
- Set $n := n + 1$ go to Start and continue.

The above determines a partial function M_D whose value $M_D(T, m)$ is the m'th (not necessarily final) element of the list L^m which is L after the m'th iteration of the meta-algorithm. Unless L^m does not have at least m elements in which case we set $M_D(T, m)$ to be undefined.

We may informally summarize the properties of M_D as follows. In what follows by "perceive" we will mean decide stably. If D perceives that some $O \neq \mathcal{G}$ is (T, T)-stable then $M = M_D$ satisfies that O is not (M, T)-stable, and so that $\mathcal{G}$ is (M, T)-stable. If D perceives that $\mathcal{G}$ is (T, T)-stable then $\mathcal{G}$ is not (M, T)-stable. If D perceives that no O is (T, T)-stable, or perceives that $T \notin \mathcal{T}^{\pm}$ then again $\mathcal{G}$ is (M, T)-stable. Finally if D does not stably decide $\mathcal{P}(T)$ then no O is (M, T)-stable. In particular we obtain:

Lemma 5.2. *If D is stably sound then $s - sound(M_D)$. Moreover, for any $T \in \mathcal{T}^{\pm}$*

$$\neg\Theta^s_{M_D,T} \vee \neg(D \text{ stably soundly decides } \mathcal{P}(T)).$$

Proof. If D is stably sound then for any $T \in \mathcal{T}^{\pm}$, by construction, some O is (M_D, T)-stable only if

$$(O = \mathcal{G}) \wedge \neg\Theta^s_{M_D,T},$$

in particular such O would have property $sC(M_D, T)$. So the first part of the lemma follows. If D stably soundly decides $\mathcal{P}(T)$ then by construction $\mathcal{G}$ is (M_D, T)-stable, so the second part also follows. □

As a consequence we have:

Theorem 5.3. *There is no computable $D \in \mathcal{D}$ that stably soundly decides $\mathcal{P}$.*

Proof. Suppose otherwise that there is such a D, then by the above lemma we obtain:

$$\forall T \in \mathcal{T}^{\pm} : \neg\Theta^s_{M_D,T},$$

but this is absurd since by construction M_D is computable if D is. □

Definition 5.4. *For $D \in \mathcal{D}$, we say that $\mathcal{A}(D)$ holds if for any $T \in \mathcal{T}^{\pm}$, whenever it is true that no O is (T, T) stable, $\hbar$ is (D, T')-stable for some T' satisfying: $T \simeq_s T'$.*

The following readily implies Theorem 1.3 as the statements $\mathcal{H}(T)$ are in the language of Turing machines and so are logically equivalent to statements in arithmetic.

Theorem 5.5. *For $D \in \mathcal{D}$ the following cannot hold simultaneously: D is stably sound, D is computable, and $\mathcal{A}_D$. In particular, for $D \in \mathcal{D}$ the following cannot hold simultaneously: D is stably sound, D is computable, and whenever the statement $\mathcal{H}(T)$:*

$$(T \in \mathcal{T}^{\pm}) \wedge (\text{no } O \text{ is } (T, T)\text{-stable})$$

is true, D stably decides $\mathcal{P}(T)$.

Proof. If D is stably sound then by Lemma 5.2 $s-sound(M)$ for $M = M_D$. If D is computable then some $T \in \mathcal{T}^{\pm}$ computes M by construction. In this case no O is (T,T)-stable since otherwise M has non-empty stable output, and by construction of M this can happen only if $\mathcal{G}$ is (M,T)-stable in which case $\neg s-sound(M)$ by Theorem 4.10. So if $\mathcal{A}(D)$ then for some T' stably computing M $\hbar$ is $D(T')$-stable and so by construction of M $\mathcal{G}$ is (M,T')-stable, but then $\mathcal{G}$ is (T,T)-stable which is a contradiction. □

6. 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, or can compute in principle what "input" our S receives it doesn't matter what kind of environment they are in. For practical purposes S has in his 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 $I \in \mathcal{I}$ as a string in his computer, after possibly using his computer in some way, S instructs his computer to print after some indeterminate time a "string":

$$D_S(I) \in \mathcal{U} \times \{\pm\} \bigsqcup \{\hbar\} \times \{\pm\} \bigsqcup \{\epsilon\}.$$

So S is meant to determine an element of $D_S \in \mathcal{D}$, which we interpret as a certain decision machine as described in the previous section.

Remark 6.1. *The above is partially a simplification, because for a real world S it may be that each $D_S(I)$ must be understood as a probability distribution on $\mathcal{U} \times \{\pm\} \bigsqcup \{\hbar\} \times \{\pm\} \bigsqcup \{\epsilon\}$. In other words the value $D_S(I)$ may only be determined up to some dice roll, which we may expect if quantum mechanics plays a significant role. This extra complexity will be ignored, as it does not meaningfully change any of our arguments since dice rolls can be simulated completely with Turing machines. Moreover, we are only interested in stable output of D_S which, given our interpretation, should not be affected by any dice rolls.*

We will say ***physical*** S when we want to clarify that we are talking of the actual human and not an abstract associated machine. In what follows when we say "perceive" we mean that our physical S eventually decides something and will never change their mind on that particular decision, formalized analogously to our definition of stably sound machines. We emphasize that although we talk of our S as a physical subject doing things like perceiving or deciding, we are just talking of various machines associated to this subject and of the mathematical properties of these machines.

In terms of our subject S, $\neg\mathcal{A}(D_S)$ in particular means that there exists a $T_S \in \mathcal{T}^{\pm}$ so that the statement $\mathcal{H}(T_S)$:

$$(T_S \in \mathcal{T}^{\pm}) \wedge (\text{no } O \text{ is } (T_S, T_S)\text{-stable})$$

is true but S will never perceive it to be true. On the other hand $\mathcal{A}(D_S)$ is of course implied by S being able to perceive that a given Turing machine

$$f : \mathbb{N} \to \mathbb{N} \times \{\pm\}$$

has no stable output if f in fact does not have stable output. However, the condition $\mathcal{A}(D_S)$ is likely weaker since for T_S as above $T_S(T_S, \cdot)$ stably computes the partial function

$$u : \mathbb{N} \to \mathcal{U} \times \{\pm\}$$

that is nowhere defined. One can then hope, since u is so simple, that T_S can be put into a normal form T' with $T_S \simeq_s T'$ and with $T'(T', \cdot) = u$ and so that S can perceive that $T'(T', \cdot) = u$.

If we substitute D_S into the statement of Theorem 5.5 and define $\mathcal{H}(S) := \mathcal{H}(T_S)$ then we obtain our Theorem 1.4.

6.1. **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 [14], [15]. Our argument for Theorem 1.4 extends Gödel's idea [12, 310], although we are 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 [15], 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 1.4. 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 there could be such a stably undecidable arithmetic statement of the form $\mathcal{H}(S)$ above. To me personally this seems unlikely precisely because stable soundness is such a loose assumption, and given my own intuition into human intelligence. However such a philosophical discussion is outside our scope.

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

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] H. B. Curry, *Functionality in combinatory logic.*, Proc. Natl. Acad. Sci. USA, 20 (1934), pp. 584–590.
[6] D. Deutsch, *Quantum theory, the Church-Turing principle and the universal quantum computer.*, Proc. R. Soc. Lond., Ser. A, 400 (1985), pp. 97–117.
[7] S. Feferman, *Are There Absolutely Unsolvable Problems? Gödel's Dichotomy*, Philosophia Mathematica, 14 (2006), pp. 134–152.
[8] C. Fields, D. Hoffman, C. Prakash, and M. Singh, *Conscious agent networks: Formal analysis and application to cognition*, Cognitive Systems Research, 47 (2017).
[9] P. Grindrod, *On human consciousness: A mathematical perspective*, Network Neuroscience, 2 (2018), pp. 23–40.
[10] 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.
[11] J.R. Lucas, *Minds machines and Gödel*, Philosophy, 36 (1961).
[12] K. Gödel, *Collected Works III (ed. S. Feferman)*, New York: Oxford University Press, 1995.
[13] A. Kent, *Quanta and qualia*, Foundations of Physics, 48 (2018), pp. 1021–1037.
[14] 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.
[15] ———, *On the question of whether the mind can be mechanized, ii: Penrose's new argument*, Journal of Philosophy, 115 (2018), pp. 453–484.
[16] K. Kremnizer and A. Ranchin, *Integrated information-induced quantum collapse*, Foundations of Physics, 45 (2015), pp. 889–899.
[17] R. Penrose, *Beyond the shadow of a doubt*, Psyche, (1996).

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