re: turing's diagonals

how to decide on the sequence of computable numbers

Nicholas Swenson
dart200@gmail.com

This paper directly refutes the motivating points of §8: Application of the diagonal process from Alan Turing's paper *On Computable Numbers*. After briefly touching upon the uncontested fact that computational machines are necessarily fully enumerable, we will discuss an alternative to Turing's algorithm for computing direct diagonal across the computable numbers. This alternative not only avoids an infinite recursion, but also any sort of decision paradox. Then, by using techniques described in §3 of how to resolve a halting paradox to correct the interface of decision machine $\mathcal{D}$, we will mitigate the decision paradox that occurs in Turing's attempt at computing a direct diagonal, and show that it still does compute a direct diagonal. Finally, we will analogously fix the decision paradox found in trying to compute an inverse diagonal, but in this case we will demonstrate that the resulting computation is not sufficient to produce a complete inverse diagonal. Opposed to Turing's several objections, there is no way to utilize a paradox-resistant correction of $\mathcal{D}$, that can actually exist, to compute an inconsistency that would make the fully enumerated sequence of computable numbers incoherent with itself. This should hopefully free us up to begin seeking out the specific algorithm $\mathcal{D}$ might actually run.

Pseudocode notes:
Function definition is derived from lambda syntax used is modern languages like kotlin/typescript:

```
        function_name = (params) -> { code }, curly braces optional
```

1: bijection onto natural numbers

By the nature of their construction Turing machines, and indeed any computer program, can be uniquely represented by a single finite-length natural number. The particular number that represents a machine is entirely dependent on the specific language that the machine description has been encoded by, but given a specific language each unique machine is described by a unique number. If one is skeptical of this please consider the fact that most all the program files you write are just strings of characters, each one or two bytes long, and put all together can be viewed as a single number. If your program is more than one file, a complete description of the system written is found in the concatenation of all files together into a single number.

Therefore, it is possible to enumerate all computer programs by stepping thru the natural numbers, testing each one for validity within the desired language. *Valid numbers are a full description of the program, and given an appropriate interpreter can be executed directly*. By throwing out numbers which don't test as valid within the language, we can biject the rest back onto the natural numbers. A side effect of this is granting us an ability to *count* or step thru all possible computer programs in a discrete manner, one by one, with the certainty we will encounter and account for all possible programs along the way, all while maintaining a persistent total order that encompasses all possible programs. This bijection/enumerability the subject of §5 from Turing's paper:

To each computable sequence there corresponds at least one description number, while to no description number does there correspond more than one computable sequence. The computable sequences and numbers are therefore enumerable

While Turing does utilize this notion in his later arguments, the full effect of this enumeration goes underutilized in §8 while arguing against enumeration algorithms for subsets of computing machines, such as computable numbers. This paper will keep the enumerability at the forefront of our thoughts while reexamining the diagonals.

The very first consideration Turing wrote about after devising the model defining the computing machines capable of computing numbers, was the potential to produce a Cantor's style inverse diagonal across all computable numbers (deemed β), something that would prove contradictory to the fact we can enumerate all machines utilizing natural numbers:

It may be thought that arguments which prove that the real numbers are not enumerable would also prove that the computable numbers and sequences cannot be enumerable

There are flaws the arguments that follow, but before we get to that let us first dive into the second diagonal that Turing considers: β' the direct diagonal formed across all computable numbers, where each Nth digit is the computed Nth digit of the Nth computable number.

2: the computable direct diagonal

Turing brings up this computation with the intention of demonstrating a decision inconsistency with a hypothetical decision machine $\mathcal{D}$, that can decide if any machine $\mathcal{M}$ is "satisfactory" in regards to being able to fully compute some computable number. Turing described this nature as being "circle-free", meaning input machine $\mathcal{M}$ would never get caught in any computational circles (or infinite loops) while computing any digit for the number it computes. While the notion of calling, inputs, or returns weren't well formalized at the time, in more modern terms: $\mathcal{D}$ would deem $\mathcal{M}$ satisfactory if for any given input natural number *n*, $\mathcal{M}$ can in finite time compute and return an *n*-th digit for the number it's computing.

```
𝒟 = (ℳ: machine) -> {
  "s"atisfactory   : if ℳ(n) returns a digit for all natural numbers n
  "u"nsatisfactory : otherwise
}
```

With such a machine $\mathcal{D}$, creating machine $\mathcal{H}$ which can compute a direct diagonal is a matter of iterating across all the natural numbers, utilizing $\mathcal{D}$ to test numbers if they indeed represent a satisfactory machine, and then executing (or simulating) the satisfactory machine to the appropriate digit:

In the N-th section [of $\mathcal{H}$] the machine $\mathcal{D}$ tests the number N. If N is satisfactory, i.e., if it is the D.N of a circle-free machine, then R(N) = 1 + R(N-1) and the first R(N) figures of the sequence of which a D.N. is N are calculated. The R(N)-th figure of this sequence is written down as one of the figures of the sequence β' computed by $\mathcal{H}$. If N is not satisfactory, then R(N) = R(N— 1) and the machine goes on to the (N+1)-th section of its motion.

Turing defined his algorithms operating in “motions”/iterations of “writing down” the computed “figures”/digits of β’. For the sake of highlighting errors, this paper will instead discuss diagonals in the form of algorithms that compute any particular consecutive r-th digit of that diagonal. One can find the full diagonal by iterating across all the natural numbers as an input:

```
𝓗 = (r) -> {
  rn = 0                      // counter for found computable numbers
  for(n = 0; true; n++) {     // iterator over the natural numbers
    if (𝒟(n) == unsatisfactory)
      continue
    elif (rn != r)            // iterate until the r-th “s” machine
      rn++
    else
      return n(r)             // language can execute description number
  }
}
```

It’s worth noting that as one iterates on the natural numbers, one will encounter more machines that compute computable numbers than actual computable numbers. Given the trivial ways one can change a state machine without affecting the final halting state/return value (such as adding any arbitrary `wait`) there are actually infinite ways to build a computing machine for any given computable number. If `Q(n)`: amount of computable numbers found by natural number n, and `R(n)`: amount of machines that compute numbers found, then `Q(n) <= R(n)` for all n. This algorithm does nothing to deduplicate those numbers within the computed diagonal, and therefore any given computable number will be represented an infinite number of times on this diagonal. Though this paper sits opposed to the notion that said deduplication is fundimentally unfeasible due to Turing equivalence being generally undecidable... this duplication is irrelevant to the claims being made, so we will simply leave the duplication in place.

Turing then argues the nature of construction of $\mathcal{H}$ (which computes β’) must be “satisfactory”:

> *From the construction of $\mathcal{H}$ we can see that $\mathcal{H}$ is circle-free. Each section of the motion of $\mathcal{H}$ comes to an end after a finite number of steps.*

The decision inconsistency Turing then considers happens when the diagonal computation encounters the machine $\mathcal{H}$, with description number K, that computes it:

> *The computation of the first R(K) — l figures would be carried out all right, but the instructions for calculating the R(K)-th would amount to "calculate the first R(K) figures computed by $\mathcal{H}$ and write down the R(K)-th". This R(K)-th figure would never be found.*

Put in terms of our pseudocode above: since all machines must certainly be enumerated by iterating thru the natural numbers, for some input r_h the associated natural number iterator will iterate on n_h [== K] the number that uniquely describes the machine $\mathcal{H}$ itself *At this point a bit of a problem occurs*: if $\mathcal{D}(n_h)$ reports the computation of $\mathcal{H}$ is “satisfactory” and circle-free, then the following call $n_h(r_h)$ executes $\mathcal{H}(r_h)$ which certainly initiates an infinite loop conflicting the decision. Therefore $\mathcal{H}$ as described cannot be “satisfactory”, and this we can all agree is clear.

Since the motivating reason for this exercise was concluding that $\mathcal{D}$ cannot exist, Turing uses this inconsistency as a final nail in the coffin for $\mathcal{D}$:

> *$\mathcal{H}$ is circular, contrary both to what we have found in the last paragraph and to the verdict "s". Thus both verdicts are impossible and we conclude that there can be no machine $\mathcal{D}$.*

There is unfortunately a critical error with this argument: **there are ways to compute β' without a decision inconsistency arising**. The fix is quite simple. The diagonal computation must know the unique natural number n_h that describes it, and when this is encountered it returns a computable value instead of trying to simulate itself in an infinite recursion. This simplest case is just returning a fixed 0 or 1, both work. They do not conflict in their computation because either return value forms a unique machine, with a unique location in the enumeration of computable numbers, granting them a unique location in the associated diagonal computation. Despite the fact there are infinite machines that compute a diagonal, these diagonal machines only need to know their *own* number, for when querying for their *own* digit on the diagonal. For digits from any other machine, including others which compute direct diagonals, they will query/execute that machine specifically for the digit, doing so only if that machine is determined as satisfactory.

```
𝓗_alt = (r) -> {
  rn = 0
  for(n = 0; true; n++) {
    if (𝓓(n) == unsatisfactory)
      continue
    elif (rn != r)
      rn++
    elif (n == nₕ)          // fix
      return 0
    else
      return n(r)
  }
}
```

What was a misguided attempt to produce an inconsistency in order to align with a preconceived notion, was actually just an inappropriate algorithm to compute the direct diagonal. We cannot, however, just ignore the computation that Turing did bring up: If $\mathcal{D}$ can exist, then $\mathcal{H}$ as first defined is still a valid computing machine that potentially could be run, so what would happen if someone tried? Unlike the inconsistency Turing tried to argue earlier, we now have to discuss a proper decision paradox: If $\mathcal{D}(n_h)$ decides that $\mathcal{H}$ is "satisfactory" then $\mathcal{H}(r_h)$ results in an unsatisfactory loop, and if $\mathcal{D}(n_h)$ decides $\mathcal{H}$ is "unsatisfactory" then $\mathcal{H}$ skips itself for the next satisfactory machine, making $\mathcal{H}$ satisfactory . This can be resolved by the techniques analogous to what is described in how to resolve a halting paradox: *the decider can only return true iff the decision will remain true to the end of its decision context, and will return false otherwise*.

Following this guidance we will align the “satisfactory” case with true. Within the call context of $\mathcal{H}(\mathtt{r_h})$: $\mathcal{D}(\mathtt{n_h})$ must return “unsatisfactory” as returning “satisfactory” will certainly lead to an unsatisfactory circle/infinite loop. $\mathcal{H}$ will then skip over itself proceeding to the next “satisfactory” machine to find that $\mathtt{r_h}$-th digit of that next machine to return on input $\mathtt{r_h}$. For any other call context, *including $\mathcal{H}$ where input r != r_h*: $\mathcal{D}(\mathtt{n_h})$ will return “satisfactory”.
Finally, one may then wonder what $\mathcal{H}$ is actually computing, if not a diagonal? It skips ever executing itself, and calls the $(\mathtt{r_h}+1)$-th machine twice, once for $\mathcal{H}(\mathtt{r_h})$ and another for $\mathcal{H}(\mathtt{r_h}+1)$. This however still does in fact compute a proper diagonal, as for example $\mathcal{H}$`_alt`$(\mathtt{r_h})$ will execute $\mathcal{H}(\mathtt{r_h})$ for its $\mathtt{r_h}$-th value. That is the only point at which the machines differ at all in method of computation, but the result is the same. Consider $\mathcal{H}$ in a lineup of five computable numbers intersecting the diagonal computation:

…	f	g	$\mathcal{H}$	i	j	…
⋱	⋮	⋮	⋮	⋮	⋮	
	f_{-2}	g_{-2}	$h_{-2} = f_{-2}$	i_{-2}	j_{-2}	
	f_{-1}	g_{-1}	$h_{-1} = g_{-1}$	i_{-1}	j_{-1}	
	f_0	g_0	$h_0 = i_0$	i_0	j_0	
	f_{+1}	g_{+1}	$h_{+1} = i_{+1}$	i_{+1}	j_{+1}	
	f_{+2}	g_{+2}	$h_{+2} = j_{+2}$	i_{+2}	j_{+2}	
	⋮	⋮	⋮	⋮	⋮	⋱

The grey boxes represents a section of the “actual” diagonal computation, but one can notice that this is equivalent to the digit sequence being computed by the column $\mathcal{H}$.

3: the impossible inverse diagonal

Let us head back to the initial claims of §8 where Turing grapples with the possibility of an inverse diagonal β, where each Nth value is the inverse of the Nth value of the Nth computable number. Turing leads with a succinct proof for why β certainly cannot be computable:

> let a_n be the n-th computable sequence, and let $\varphi_n(m)$ be the m-th figure in a_n. Let β be the sequence with $1-\varphi_n(m)$ as its n-th. figure. Since β is computable, there exists a number K [== β] such that $1-\varphi_n(n) = \varphi_K(n)$ for all n. Putting n = K, we have $1 = 2\varphi_K(K)$, i.e. 1 is even. This is impossible.

Essentially this is demonstrating that if β was computable it would eventually encounter itself, and have to give the inverse to a digit from its own computation, and that doing so is equivalent to trying to make 1 even. This we can all agree is true, clearly β cannot be computable, and therefore we must have an underlying reason for why. Turing first considers perhaps:

> The computable sequences are therefore not enumerable

This of course runs contrary to the notion that all Turing machines are enumerable, and Turing quickly shuts it down with an alternative proposal:

> It would be true if we could enumerate the computable sequences by finite means, but the problem of enumerating computable sequences is equivalent to the problem of finding out whether a given number is the D.N of a circle-free machine, and we have no general process for doing this in a finite number of steps

Turing's alternative proposal is that while computable numbers are enumerable, we must not have an actual finite runtime algorithm to do so, lest β would be computable. Turing does have some misgivings with this argument as a sole reasoning:

> This proof, although perfectly sound, has the disadvantage that it may leave the reader with a feeling that "there must be something wrong"

Despite this, he powers onto the next page considering β' and a proof attempt that successfully satiates any misgivings he must have felt. We have in this paper, however, already thoroughly analyzed and refuted such part of his argument in the previous section. And so where does that leave us with β? If $\mathcal{D}$ cannot be concluded as nonexistent thru analysis of computing β', does that mean β actually is computable? No, this paper fully agrees with Turing that β cannot actually be computed by a defined algorithm, and will demonstrate this regardless of $\mathcal{D}$'s existance. To discover why, let us consider the machine $\mathcal{I}$ that might compute β:

```
𝓘 = (r) -> {
  rn = 0
  for(n = 0; true; n++) {
    if (𝓓(n) == unsatisfactory)
      continue
    elif (rn != r)
      rn++
    else
      return 1-n(r)          // returns the inverse digit
  }
}
```

This machine has an identical form to $\mathcal{H}$ except for the final return statement returning the inverse digit for the associated computable number. The machine then does involve the same kind of programmatic looping issues as we found in $\mathcal{H}$: at some r_i where n_i corresponds to the unique number that describes $\mathcal{I}$ if $\mathcal{D}(n_i)$ returns "satisfactory", then $1-n(r_i)$ will be executing $1-\mathcal{I}(r_i)$ in an infinite recursive loop.

First, can $\mathcal{I}$ be fixed to avoid the paradox by adding a computable/static return value short cutting the problematic recursion? Well this can avoid the paradox, but it does not fix our inverse computation. If this was tried, $\mathcal{I}$ would return this value for both its own "inverse" computation, as well as the direct diagonal iterating across it, which would make that value it's direct digit, not an inverse digit. There is in fact no method by which an inverse diagonal could determine and return a digit inverse to what it actually does return ... such a concept is pure nonsense from a logical point of view. Any digit which it does return becomes its direct value, not an inverse value.

At this point it is clear that $\mathcal{I}$ cannot be fixed to fully compute a β. There is no actual coherent method which could be used to compute β, even with $\mathcal{D}$, and we could stop here. But one may still be left wondering what happens if we just run $\mathcal{I}$ as is, with the decision paradox seemingly left in place? Once the decider $\mathcal{D}$ is fixed in regards handling the paradox, and analogously to $\mathcal{H}$, when the input to $\mathcal{I}$ is r_i then $\mathcal{D}(n_i)$ will respond "unsatisfactory". This causes $\mathcal{I}$ to skip over itself (*only on input* r_i) for the next "satisfactory" machine, and return a digit inverse of that next machine. Machine $\mathcal{I}$ produces a partial inverse diagonal that skips inverting itself. That single skip is enough to allow the direct diagonal to still reference and compute its digit, meaning $\mathcal{I}$ is not successful in computing a β that cannot be listed in the full sequence of computable numbers. Consider this table showing $\mathcal{I}$ in the lineup of computable numbers:

…	g	h	$\mathcal{I}$	j	k	…
⋱	⋮	⋮	⋮	⋮	⋮	
	g_{-2}	h_{-2}	$i_{-2} = 1-g_{-2}$	j_{-2}	k_{-2}	
	g_{-1}	h_{-1}	$i_{-1} = 1-h_{-1}$	j_{-1}	k_{-1}	
	g_0	h_0	$i_0 = 1-j_0$	j_0	k_0	
	g_{+1}	h_{+1}	$i_{+1} = 1-j_{+1}$	j_{+1}	k_{+1}	
	g_{+2}	h_{+2}	$i_{+2} = 1-k_{+2}$	j_{+2}	k_{+2}	
	⋮	⋮	⋮	⋮	⋮	⋱

The $\mathcal{I}$ column corresponds to the digit outputs from the attempt at computing an inverse diagonal. Columns of preceding letters are digits from preceding computable numbers, whereas the proceeding letters are those that come after. The letters are not meant to represent some sort of alphanumeric indication of position in the total ordering of computable numbers, ei g is *not* the 7th computable number in a total order, they are only relative positions to $\mathcal{I}$. The diagonal of grey boxes represents the direct diagonal, computed by any of the infinite direct diagonal machines including $\mathcal{H}$. Tho perhaps not impossible, a direct diagonal machine is not intended to be part of the lineup being tabled.

The cell i_0 corresponds to the output for input r_i where $\mathcal{I}$ comes across the potential problem of having to inverse it's own digit. Since $\mathcal{I}$ follows the same ordering and procession of computable numbers as the direct diagonal, input r_i is also the intersection between $\mathcal{I}$ and the direct diagonal, such that the direct diagonal calls $\mathcal{I}(r_i)$ to obtain the value of digit i_0. Due to the fixed decider, $\mathcal{I}$ skips itself and subsequently returns the inverse of $j(r_i)$, computing the digit i_0 to have value $1-j_0$. **No contradiction has been computed here**.

By observing this table we can also deduce the same proof that Turing led off §8 with. If $\mathcal{I}$ attempted to set i_0 to it's the inverse of i_0, then $i_0=1-i_0$ would imply that $2i_0=1$, or that $i_0=½$, neither of which are acceptable results for the digit output of a computable number.

β is surely not a computable number, **and we don't need to throw out deciders in order to demonstrate that lack of computability.**