on ‘computability’

define

function *f*: (bool) -> bool, such that:

f(TRUE) = TRUE
f(FALSE) = FALSE

assume

f is computable, and that program F has been written:

```
bool F(bool b) {
      return b;
}
```

define

natural program D such that:

```
bool D() {
      if F(D())
            return FALSE;
      else
            return TRUE;
}
```

argue

if F(D()) is TRUE, then D() is FALSE, so F(D()) should have been FALSE
if F(D()) is FALSE, then D() is TRUE, so F(D()) should have been TRUE
therefore we have a guaranteed, pre-determinable inconsistency

conclude

program F does not exist
function *f* is not computable

now, do you really think our function *h* is uncomputable? it’s literally just a pass through function that maps the boolean argument to itself, there are only two simpler input->output mappings that exist, the two functions that map both boolean inputs to one output. it seems completely absurd to make the claim that such a program isn’t computable, because if this isn’t computable, what meaning does the definition of computability actually have?

see, the problem with allowing such a paradox as an uncomputability proof, is that we could construct an analogous proof, for *any* partially terminating program with multiple return values that are deterministically selected by the input. all we need to construct the paradox are two different outputs, respectively selected deterministically, by two different input, to devise a corresponding uncomputability proof by paradox

and so Turing did, to probably the *first* novel computation-related question he thought of:

> *does this program halt?*

let's compare the paradox to Turing's halting 'problem', *a foundational problem defining major limits to computing theory*:

define

> function *halts*: (string) -> bool, where string is the text of a program, such that when *halts* is applied to the text, and that text represents a terminating program, the result is TRUE, and if that text represents a non-terminating program, the result is FALSE.
>
> examples:
>
> *halts*(`'foo() return'`) = TRUE
> *halts*(`'bar() loop_forever'`) = FALSE

assume

> *halts* is computable, and that program H has been written:

```
bool Halts(string program) {
      if (/* program represents a terminating program */)
            return TRUE
      else // program represents a non-terminated program
            return FALSE
}
```

define

> natural program D such that:

```
D() {
      if Halts(D)
            loop_forever
      else
            return
}
```

argue

> if `Halts(D)` is TRUE, then `D()` is non-terminating, so `Halts(D)` should have been FALSE
> if `Halts(D)` is FALSE, then `D()` is terminating, so `Halts(D)` should have been TRUE
> therefore we have a guaranteed, pre-determinable inconsistency

conclude

> program H does not exist
> function *halts* is not computable

and so, by an analogous form of constructed paradox, *halts* is deemed uncomputable, and a computational device cannot objectively compute if another program halts, or not. i guess.

but, by this point, i'm not sure what computability even means. is *almost* nothing computable in theory? are humans just being delusional, in 'theory', when they sit down to write meaningful software?

… an interesting *corollary* of this paradox, is that the halting oracle (*halts*) can actually know what will happen. by the nature of the constructed paradox, it's actually given **the choice** to decide whether program D will terminate or not, as D is a-priori guaranteed to terminate if the oracle returns FALSE, or run indefinitely if the oracle returns TRUE. *the halting oracle can actually decide and know what will happen*, ***it just can't return that information to D***, while maintaining its choice in tact.

but it could, for example, send that knowledge to a 3rd, unaffiliated party with no influence on D, and then lie to D, to get the result it told that 3rd, unaffiliated party ... such as you.

reference

Alan M. Turing, *on Computable Numbers with an Application to the Entscheidungsproblem*, 1937

Eric C.R. Hehner, *Problems with the Halting Problem*, 2013