# Refutation of Halting Problem Diagonalization Argument

```
#define u32 uint32_t


int Simulate(u32 P, u32 I)
{
  ((void(*)(u32))P)(I);
  return 1;
}


int D(u32 P)    // P is a machine address
{
  if ( H(P, P) )
    return 0    // reject when H accepts
  return 1;     // accept when H rejects
}


int main()
{
  H((u32)D, (u32)D);
}
```

We can know that simulating halt decider H must stop simulating its input because if H did not stop simulating its input then D would have the same halting behavior as if D called Simulate instead of H.

The above analysis is confirmed by actual execution of the above function in the x86utm operating system. H detects an infinitely repeating non-halting pattern that never reaches the second line of D.

X86utm was designed so that halting problem computations can be examined concretely at the high level of abstraction of the C programming language. The x86utm operating system provides a DebugStep() function to allow any C function to execute the x86 machine language of another C function in debug step mode. Because these C functions are executed in separate process contexts they do not interfere with each other.

The partial halt decider H invokes an x86 emulator to execute its input D in debug step mode. The input is the machine address of the input x86 function cast to a 32-bit unsigned integer.

H examines the complete execution trace of D immediately after each x86 instruction of D is simulated. As soon as the partial halt decider H recognizes a non-terminating behavior pattern of D it aborts the simulation of D and reports not-halting.

When H is a simulating halt decider H(D,D) rejects its input as a halting computation on the basis that H(D,D) specifies infinitely nested simulation to H unless H aborts its simulation of D(D).

**Table T (All Turing machines on each other as input)**

| | <M1> | <M2> | <M3>... | <D>... |
|---|---|---|---|---|
| M1 | accept | | | reject |
| M2 | | reject | | accept |
| M3 | | | ~halt | accept |
| ... | | | | |
| D | reject | accept | accept | accept |
| ... | | | | |

Table TH is defined on the basis of Table T where:
(a) accept becomes accept (b) reject becomes reject (c) ~Halt becomes reject

**Table TH (Turing machine H on all Turing Machine pairs as input)**

| | <M1> | <M2> | <M3>... | <D>... |
|---|---|---|---|---|
| M1 | accept | | | reject |
| M2 | | reject | | accept |
| M3 | | | reject | accept |
| ... | | | | |
| D | reject | accept | accept | reject |
| ... | | | | |

On the diagonal: a TM is executed with its own TM description as input. Table TD only has a single input that reverses the value of the diagonal of table TH for each TM description on the horizontal axis of table TH.

**Table TD (reverses H decision along the diagonal of table TH)**

| <M1> | <M2> | <M3>... | <D>... |
|---|---|---|---|
| reject | accept | accept | accept |

**All of the table values are correct.**
All of the values in TD must be the opposite of the values of the TH diagonal is satisfied.
The reject value of table TH at (D, <D>) corresponds to the actual behavior of H(D,<D>).
The accept value of table T at (D, <D>) corresponds to the actual behavior of D(<D>) also shown at element <D> in table TD.

Because the requirement that table TH have the same (accept / reject) value as table T directly contradicts the actual behavior of H(D,<D>) and D(<D>) we can toss out this requirement as erroneous.

It is a self contradictory requirement. It requires table T(D,<D>) to have the same value as table TD(<D>) (that is fine). It requires the diagonal of TH to have the opposite value as table TD (that is fine). When we add the requirement that one element of the diagonal have the same value as the same element of TH we directly contradict the prior requirements.

When-so-ever there is a requirement for an object to have simultaneous mutually exclusive properties then this requirement is necessarily erroneous.



**Sipser, Michael 1997.** Introduction to the Theory of Computation. Boston: PWS Publishing Company (165-167)