# Refuting the Sipser Halting Problem Diagonalization Argument

Every machine that halts in a reject state is a halting computation. When machine D is inserted into Figure 4.5 deriving Figure 4.6 the fact that a reject state is a halting computation is ignored. This makes the values at $\langle D, \langle M_1 \rangle \rangle$ and $\langle D, \langle M_1 \rangle \rangle$ in Figure 4.6 incorrect. When machine D is inserted into both Figure 4.4 and Figure 4.5 correctly (figures 4.4b and 4.5a respectively) the contradiction is eliminated.

| | $\langle M_1 \rangle$ | $\langle M_2 \rangle$ | $\langle M_3 \rangle$ | $\langle M_4 \rangle$ ... |
|---|---|---|---|---|
| $M_1$ | accept | | accept | |
| $M_2$ | accept | accept | accept | accept |
| $M_3$ | | | | |
| $M_4$ | accept | accept | | |
| ... | | | | |

**Original Figure 4.4**

| | $\langle M_1 \rangle$ | $\langle M_2 \rangle$ | $\langle M_3 \rangle$ | $\langle M_4 \rangle$ ... |
|---|---|---|---|---|
| $M_1$ | accept | ~halt | accept | ~halt |
| $M_2$ | accept | accept | accept | accept |
| $M_3$ | ~halt | ~halt | ~halt | ~halt |
| $M_4$ | accept | accept | ~halt | ~halt |
| ... | | | | |

**Figure 4.4a** (converted from Figure 4.4 making ~halt assumption explicit)

| | $\langle M_1 \rangle$ | $\langle M_2 \rangle$ | $\langle M_3 \rangle$ | $\langle M_4 \rangle$ ... |
|---|---|---|---|---|
| $M_1$ | accept | reject | accept | reject |
| $M_2$ | accept | accept | accept | accept |
| $M_3$ | reject | reject | reject | reject |
| $M_4$ | accept | accept | reject | reject |
| ... | | | | |

**Original Figure 4.5** (underlining added)

| | $\langle M_1 \rangle$ | $\langle M_2 \rangle$ | $\langle M_3 \rangle$ | $\langle M_4 \rangle$ ... | $\langle D \rangle$ ... |
|---|---|---|---|---|---|
| $M_1$ | accept | ~halt | accept | ~halt | DC |
| $M_2$ | accept | accept | accept | accept | DC |
| $M_3$ | ~halt | ~halt | ~halt | ~halt | DC |
| $M_4$ | accept | accept | ~halt | ~halt | DC |
| ... | | | | | |
| D | reject | reject | accept | accept | reject |
| ... | | | | | |

**Figure 4.4b** (Insert D into Figure 4.4a)

| | $\langle M_1 \rangle$ | $\langle M_2 \rangle$ | $\langle M_3 \rangle$ | $\langle M_4 \rangle$ ... | $\langle D \rangle$ ... |
|---|---|---|---|---|---|
| $M_1$ | accept | reject | accept | reject | DC |
| $M_2$ | accept | accept | accept | accept | DC |
| $M_3$ | reject | reject | reject | reject | DC |
| $M_4$ | accept | accept | reject | reject | DC |
| ... | | | | | |
| D | accept | accept | accept | accept | accept |
| ... | | | | | |

**Figure 4.5a** (Insert D into Figure 4.5)

Where is the diagonalization in the proof of Theorem 4.9? It becomes apparent when you examine tables of behavior for TMs $H$ and $D$. In these tables we list all TMs down the rows, $M_1$, $M_2$, ... and all their descriptions across the columns, $\langle M_1 \rangle$, $\langle M_2 \rangle$, ... The entries tell whether the machine in a given row accepts the input in a given column. The entry is *accept* if the machine accepts the input but is blank if it rejects or loops on that input. We made up the entries in the following figure to illustrate the idea.

| | $\langle M_1 \rangle$ | $\langle M_2 \rangle$ | $\langle M_3 \rangle$ | $\langle M_4 \rangle$ | $\cdots$ |
|---|---|---|---|---|---|
| $M_1$ | *accept* | | *accept* | | |
| $M_2$ | *accept* | *accept* | *accept* | *accept* | |
| $M_3$ | | | | | $\cdots$ |
| $M_4$ | *accept* | *accept* | | | |
| $\vdots$ | | $\vdots$ | | | |

**FIGURE 4.4**
Entry $i, j$ is *accept* if $M_i$ accepts $\langle M_j \rangle$

In the following figure the entries are the results of running $H$ on inputs corresponding to Figure 4.4. So if $M_3$ does not accept input $\langle M_2 \rangle$, the entry for row $M_3$ and column $\langle M_2 \rangle$ is *reject* because $H$ rejects input $\langle M_3, \langle M_2 \rangle \rangle$.

| | $\langle M_1 \rangle$ | $\langle M_2 \rangle$ | $\langle M_3 \rangle$ | $\langle M_4 \rangle$ | $\cdots$ |
|---|---|---|---|---|---|
| $M_1$ | *accept* | *reject* | *accept* | *reject* | |
| $M_2$ | *accept* | *accept* | *accept* | *accept* | $\cdots$ |
| $M_3$ | *reject* | *reject* | *reject* | *reject* | |
| $M_4$ | *accept* | *accept* | *reject* | *reject* | |
| $\vdots$ | | $\vdots$ | | | |

**FIGURE 4.5**
Entry $i, j$ is the value of $H$ on input $\langle M_i, \langle M_j \rangle \rangle$

In the following figure, we added D to Figure 4.5. By our assumption, $H$ is a TM and so is $D$. Therefore it must occur on the list $M_1$, $M_2$, ... of all TMs. Note that $D$ computes the opposite of the diagonal entries. The contradiction occurs at the point of the question mark where the entry must be the opposite of itself.



| | $\langle M_1 \rangle$ | $\langle M_2 \rangle$ | $\langle M_3 \rangle$ | $\langle M_4 \rangle$ | $\cdots$ | $\langle D \rangle$ | $\cdots$ |
|---|---|---|---|---|---|---|---|
| $M_1$ | <u>*accept*</u> | *reject* | *accept* | *reject* | | *accept* | |
| $M_2$ | *accept* | <u>*accept*</u> | *accept* | *accept* | $\cdots$ | *accept* | $\cdots$ |
| $M_3$ | *reject* | *reject* | <u>*reject*</u> | *reject* | | *reject* | |
| $M_4$ | *accept* | *accept* | *reject* | <u>*reject*</u> | | *accept* | |
| $\vdots$ | | $\vdots$ | | | $\ddots$ | | |
| $D$ | *reject* | *reject* | *accept* | *accept* | | <u>?</u> | |
| $\vdots$ | | $\vdots$ | | | | | $\ddots$ |

**FIGURE 4.6**
If $D$ is in the figure, a contradiction occurs at "?"

The above portions of pages 166-167 are directly relevant to the rebuttal. (fair use)
**Sipser, Michael 1997.** Introduction to the Theory of Computation. Boston: PWS Publishing

# Appendix

```
#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.

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).