Rebutting the Sipser Halting Problem Proof V2

A simulating halt decider correctly predicts what the behavior of its input would be if this simulated input never had its simulation aborted. It does this by correctly recognizing several non-halting behavior patterns in a finite number of steps of correct simulation.

When simulating halt decider H correctly predicts that the conventional counter-example input to the halting theorem never halts (because it remains stuck in recursive simulation) this input ceases to prove the halting theorem.

MIT Professor Michael Sipser has agreed that the following verbatim paragraph is correct (he has not agreed to anything else in this paper):

(a) If simulating halt decider H correctly simulates its input D until H correctly determines that its simulated D would never stop running unless aborted then (b) H can abort its simulation of D and correctly report that D specifies a non-halting sequence of configurations.

We start with Sipser's definitions of H and D:

> On input (M, w), where M is a TM and w is a string, H halts and accepts if M accepts w. Furthermore, H halts and rejects if M fails to accept w.
> In other words, we assume that H is a TM, where
>
> H(⟨M,w⟩ = { accept if M accepts w
> { reject if M does not accept w
>
> Now we construct a new Turing machine D with H as a subroutine. This new TM calls H to determine what M does when the input to M is its own description ⟨M⟩. Once D has determined this information, it does the opposite. That is, it rejects if M accepts and accepts if M does not accept.
>
> D(⟨M⟩) = { accept if M does not accept ⟨M⟩
> { reject if M accepts ⟨M⟩ (Sipser 1997:165)

We encode the Sipser D and define the behavior of Sipser H as C functions.

```
int Sipser_D(int (*M)())
{
  if ( HH(M, M) )
    return 0;
  return 1;
}

int main()
{
  Sipser_D(Sipser_D);
}
```

H returns 0 to Sipser_D on the basis that Sipser_D correctly simulated by H would remain stuck in recursive simulation unless H aborts its simulation of Sipser_D.

Diagonal proof: Correctly predict what I will say when I will always say the opposite of whatever you predict. If you correctly predict that I will say nothing then this too is correct.

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

```
_Sipser_D()
[00001e84] 55              push ebp
[00001e85] 8bec            mov ebp,esp
[00001e87] 8b4508          mov eax,[ebp+08]
[00001e8a] 50              push eax
[00001e8b] 8b4d08          mov ecx,[ebp+08]
[00001e8e] 51              push ecx
[00001e8f] e8a0f5ffff      call 00001434
[00001e94] 83c408          add esp,+08
[00001e97] 85c0            test eax,eax
[00001e99] 7404            jz 00001e9f
[00001e9b] 33c0            xor eax,eax
[00001e9d] eb05            jmp 00001ea4
[00001e9f] b801000000      mov eax,00000001
[00001ea4] 5d              pop ebp
[00001ea5] c3              ret
Size in bytes:(0034) [00001ea5]

_main()
[00001eb4] 55              push ebp
[00001eb5] 8bec            mov ebp,esp
[00001eb7] 68841e0000      push 00001e84
[00001ebc] e8c3ffffff      call 00001e84
[00001ec1] 83c404          add esp,+04
[00001ec4] 33c0            xor eax,eax
[00001ec6] 5d              pop ebp
[00001ec7] c3              ret
Size in bytes:(0020) [00001ec7]

 machine   stack     stack     machine    assembly
 address   address   data      code       language
 ========  ========  ========  =========  ==============
[00001eb4][00103244][00000000] 55         push ebp
[00001eb5][00103244][00000000] 8bec       mov ebp,esp
[00001eb7][00103240][00001e84] 68841e0000 push 00001e84    // push Sipser_D
[00001ebc][0010323c][00001ec1] e8c3ffffff call 00001e84    // call Sipser_D
[00001e84][00103238][00103244] 55         push ebp
[00001e85][00103238][00103244] 8bec       mov ebp,esp
[00001e87][00103238][00103244] 8b4508     mov eax,[ebp+08] // move arg to eax
[00001e8a][00103234][00001e84] 50         push eax         // push Sipser_D
[00001e8b][00103234][00001e84] 8b4d08     mov ecx,[ebp+08] // move arg to ecx
[00001e8e][00103230][00001e84] 51         push ecx         // push Sipser_D
[00001e8f][0010322c][00001e94] e8a0f5ffff call 00001434    // call H
New slave_stack at:1032e8

Begin Local Halt Decider Simulation   Execution Trace Stored at:1132f0
[00001e84][001132dc][001132e0] 55         push ebp         // begin Sipser_D
[00001e85][001132dc][001132e0] 8bec       mov ebp,esp
[00001e87][001132dc][001132e0] 8b4508     mov eax,[ebp+08] // move arg to eax
[00001e8a][001132d8][00001e84] 50         push eax         // push Sipser_D
[00001e8b][001132d8][00001e84] 8b4d08     mov ecx,[ebp+08] // move arg to ecx
[00001e8e][001132d4][00001e84] 51         push ecx         // push Sipser_D
[00001e8f][001132d0][00001e94] e8a0f5ffff call 00001434    // call H
New slave_stack at:14dd10
[00001e84][0015dd04][0015dd08] 55         push ebp         // begin Sipser_D
[00001e85][0015dd04][0015dd08] 8bec       mov ebp,esp
[00001e87][0015dd04][0015dd08] 8b4508     mov eax,[ebp+08] // move arg to eax
[00001e8a][0015dd00][00001e84] 50         push eax         // push Sipser_D
[00001e8b][0015dd00][00001e84] 8b4d08     mov ecx,[ebp+08] // move arg to ecx
[00001e8e][0015dcfc][00001e84] 51         push ecx         // push Sipser_D
[00001e8f][0015dcf8][00001e94] e8a0f5ffff call 00001434    // call H
Local Halt Decider: Infinite Recursion Detected Simulation Stopped
```

Sipser_D(Sipser_D) remains stuck in recursive simulation until H recognizes this and aborts its simulation. The above behavior pattern conclusively proves that Sipser_D(Sipser_D) cannot possibly stop running unless H aborts its simulation of Sipser_D.