Simulating Halt Decider Applied to the Halting Theorem

The novel concept of a simulating halt decider enables halt decider H to to correctly determine the halt status of the conventional “impossible” input D that does the opposite of whatever H decides. This works equally well for Turing machines and “C” functions. The algorithm is demonstrated using “C” functions because all of the details can be shown at this high level of abstraction.

Simulating halt decider H correctly determines that D correctly simulated by H would remain stuck in recursive simulation never reaching its own final state. D cannot do the opposite of the return value from H because this return value is unreachable by every correctly simulated D. This same result is shown to be derived in the Peter Linz Turing machine based proof.

> In computability theory, the halting problem is the problem of determining, from a description of an arbitrary computer program and an input, whether the program will finish running, or continue to run forever. Alan Turing proved in 1936 that a general algorithm to solve the halting problem for all possible program-input pairs cannot exist.
>
> For any program H that might determine if programs halt, a "pathological" program D, called with some input, can pass its own source and its input to H and then specifically do the opposite of what H predicts D will do. No H can exist that handles this case. https://en.wikipedia.org/wiki/Halting_problem

```
int D(int (*x)())
{
  int Halt_Status = H(x, x);
  if (Halt_Status)
    HERE: goto HERE;
  return Halt_Status;
}

int main()
{
  Output("Input_Halts = ", H(D,D));
}
```

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

> (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.

The above words are a tautology in that the meaning of the words proves that they are true: (b) is a necessary consequence of (a). {never stop running unless aborted} is equivalent to {would never reach is own “return” instruction final state} thus never halts.

Because every decider must compute the mapping from its input to an accept or reject state and H does correctly determine that D correctly simulated by H would never halt then H is necessarily correct to reject D as non-halting.

In the hypothetical case where H never aborts the simulation of its input D(D) and H(D,D) and H1(D,D) never halt conclusively proving that H(D,D) must abort the simulation of its input and is necessarily correct for H to reject this input as non-halting.

When H correctly simulates D it finds that D remains stuck in recursive simulation
(a) D calls H that simulates D with an x86 emulator
(b) that calls a simulated H that simulates D with an x86 emulator
(c) that calls a simulated H that simulates D with an x86 emulator ...
Until the executed H recognizes this repeating state, aborts its simulation of D and returns 0.

The first page of the Appendix has all of the details about this.
A simulating halt decider computes the mapping from its input finite strings to an accept or reject state on the basis of the actual behavior specified by this input as measured by its correct simulation of this input.

Simulating halt decider H recognizes instances of recursive simulation using the same criteria that it uses in its dynamic behavior pattern that recognizes infinite recursion:

```
void Infinite_Recursion(u32 N)
{
  Infinite_Recursion(N);
}

_Infinite_Recursion()
[000013fa] 55              push ebp
[000013fb] 8bec            mov ebp,esp
[000013fd] 8b4508          mov eax,[ebp+08]
[00001400] 50              push eax
[00001401] e8f4ffffff      call 000013fa
[00001406] 83c404          add esp,+04
[00001409] 5d              pop ebp
[0000140a] c3              ret
Size in bytes:(0017) [0000140a]
```

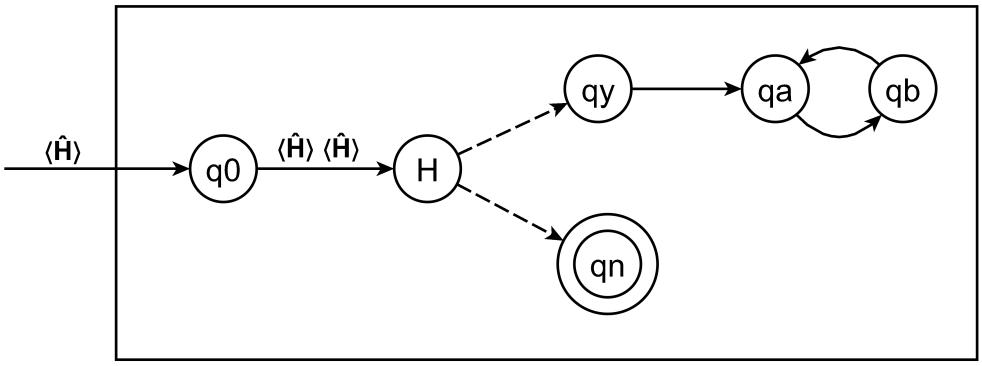
H detects that _Infinite_Recursion() calls itself with no conditional branch instructions between the beginning of _Infinite_Recursion() and the call to itself that could escape repeated recursion.

Complete halt deciding system (Visual Studio Project)
(a) x86utm operating system
(b) x86 emulator adapted from libx86emu to compile under Windows
(c) Several halt deciders and their sample inputs contained within Halt7.c
(d) The execution trace of H applied to D is shown in Halt7out.txt
https://liarparadox.org/2023_02_07.zip

Peter Linz Halting Problem Proof adapted to use a simulating halt decider

When we see the notion of a simulating halt decider applied to the embedded copy of Linz H within Linz Ĥ then we can see that the ⟨Ĥ⟩ ⟨Ĥ⟩ input to embedded H specifies recursive simulation that never reaches its final state of ⟨Ĥ.qn⟩.

computation that halts … the Turing machine will halt whenever it enters a final state. (Linz:1990:234)



Ĥ.q_0 ⟨Ĥ⟩ ⊢* embedded_H ⟨Ĥ⟩ ⟨Ĥ⟩ ⊢* Ĥ.qy ∞
If ⟨Ĥ⟩ ⟨Ĥ⟩ correctly simulated by embedded_H would reach its own final state of ⟨Ĥ.qn⟩.

Ĥ.q_0 ⟨Ĥ⟩ ⊢* embedded_H ⟨Ĥ⟩ ⟨Ĥ⟩ ⊢* Ĥ.qn
If ⟨Ĥ⟩ ⟨Ĥ⟩ correctly simulated by embedded_H would never reach its own final state of ⟨Ĥ.qn⟩.

When Ĥ is applied to ⟨Ĥ⟩ // subscripts indicate unique finite strings
Ĥ copies its input ⟨$Ĥ_0$⟩ to ⟨$Ĥ_1$⟩ then H simulates ⟨$Ĥ_0$⟩ ⟨$Ĥ_1$⟩

Then these steps would keep repeating: (unless their simulation is aborted)
$Ĥ_0$ copies its input ⟨$Ĥ_1$⟩ to ⟨$Ĥ_2$⟩ then embedded_H_0 simulates ⟨$Ĥ_1$⟩ ⟨$Ĥ_2$⟩
$Ĥ_1$ copies its input ⟨$Ĥ_2$⟩ to ⟨$Ĥ_3$⟩ then embedded_H_1 simulates ⟨$Ĥ_2$⟩ ⟨$Ĥ_3$⟩
$Ĥ_2$ copies its input ⟨$Ĥ_3$⟩ to ⟨$Ĥ_4$⟩ then embedded_H_2 simulates ⟨$Ĥ_3$⟩ ⟨$Ĥ_4$⟩...

Since we can see that the input: ⟨$Ĥ_0$⟩ ⟨$Ĥ_1$⟩ correctly simulated by embedded_H would never reach its own final state of ⟨$Ĥ_0$.qn⟩ we know that ⟨$Ĥ_0$⟩ specifies a non-halting sequence of configurations.

Linz, Peter 1990. An Introduction to Formal Languages and Automata. Lexington/Toronto: D. C. Heath and Company. (317-320)

Appendix

When H correctly simulates D it finds that D remains stuck in recursive simulation

```
int D(int (*x)())
{
  int Halt_Status = H(x, x);
  if (Halt_Status)   // This code is never reached on D(D)
    HERE: goto HERE; // This code is never reached on D(D)
  return Halt_Status;
}

int main()
{
  Output("Input_Halts = ", H(D,D));
}

_D()
[00001d12] 55              push ebp
[00001d13] 8bec            mov ebp,esp
[00001d15] 51              push ecx
[00001d16] 8b4508          mov eax,[ebp+08]
[00001d19] 50              push eax
[00001d1a] 8b4d08          mov ecx,[ebp+08]
[00001d1d] 51              push ecx
[00001d1e] e83ff8ffff      call 00001562
[00001d23] 83c408          add esp,+08
[00001d26] 8945fc          mov [ebp-04],eax
[00001d29] 837dfc00        cmp dword [ebp-04],+00
[00001d2d] 7402            jz 00001d31
[00001d2f] ebfe            jmp 00001d2f
[00001d31] 8b45fc          mov eax,[ebp-04]
[00001d34] 8be5            mov esp,ebp
[00001d36] 5d              pop ebp
[00001d37] c3              ret
Size in bytes:(0038) [00001d37]

_main()
[00001d72] 55              push ebp
[00001d73] 8bec            mov ebp,esp
[00001d75] 68121d0000      push 00001d12
[00001d7a] 68121d0000      push 00001d12
[00001d7f] e8def7ffff      call 00001562
[00001d84] 83c408          add esp,+08
[00001d87] 50              push eax
[00001d88] 6883070000      push 00000783
[00001d8d] e810eaffff      call 000007a2
[00001d92] 83c408          add esp,+08
[00001d95] 33c0            xor eax,eax
[00001d97] 5d              pop ebp
[00001d98] c3              ret
Size in bytes:(0039) [00001d98]

 machine   stack     stack     machine    assembly
 address   address   data      code       language
 ========  ========  ========  =========  =============
[00001d72][0010305d][00000000] 55         push ebp
[00001d73][0010305d][00000000] 8bec       mov ebp,esp
[00001d75][00103059][00001d12] 68121d0000 push 00001d12
[00001d7a][00103055][00001d12] 68121d0000 push 00001d12
[00001d7f][00103051][00001d84] e8def7ffff call 00001562
```

```
H: Begin Simulation   Execution Trace Stored at:113109
Address_of_H:1562
[00001d12][001130f5][001130f9] 55         push ebp       ; begin D
[00001d13][001130f5][001130f9] 8bec       mov ebp,esp
[00001d15][001130f1][001030c5] 51         push ecx
[00001d16][001130f1][001030c5] 8b4508     mov eax,[ebp+08]
[00001d19][001130ed][00001d12] 50         push eax       ; push address of D
[00001d1a][001130ed][00001d12] 8b4d08     mov ecx,[ebp+08]
[00001d1d][001130e9][00001d12] 51         push ecx       ; push address of D
[00001d1e][001130e5][00001d23] e83ff8ffff call 00001562  ; call H
H: Infinitely Recursive Simulation Detected Simulation Stopped
```

We can see that the first seven instructions of D simulated by H precisely match the first seven instructions of the x86 source-code of D. This conclusively proves that these instructions were simulated correctly.

Anyone sufficiently technically competent in the x86 programming language will agree that the above execution trace of D simulated by H shows that D will never stop running unless H aborts its simulation of D.

H detects that D is calling itself with the exact same arguments that H was called with and there are no conditional branch instructions from the beginning of D to its call to H that can possibly escape the repetition of this recursive simulation.

```
[00001d84][0010305d][00000000] 83c408         add esp,+08
[00001d87][00103059][00000000] 50             push eax
[00001d88][00103055][00000783] 6883070000     push 00000783
[00001d8d][00103055][00000783] e810eaffff     call 000007a2
Input_Halts = 0
[00001d92][0010305d][00000000] 83c408         add esp,+08
[00001d95][0010305d][00000000] 33c0           xor eax,eax
[00001d97][00103061][00000018] 5d             pop ebp
[00001d98][00103065][00000000] c3             ret
Number of Instructions Executed(975) == 15 Pages
```

When D(D) is directly executed it halts (same as Sipser proof)

```
int D(int (*x)())
{
  int Halt_Status = H(x, x);
  if (Halt_Status)
    HERE: goto HERE;
  return Halt_Status;
}

int main()
{
  Output("Input_Halts = ", D(D));
}
```

```
_D()
[00001d12] 55              push ebp
[00001d13] 8bec            mov ebp,esp
[00001d15] 51              push ecx
[00001d16] 8b4508          mov eax,[ebp+08]
[00001d19] 50              push eax
[00001d1a] 8b4d08          mov ecx,[ebp+08]
[00001d1d] 51              push ecx
[00001d1e] e83ff8ffff      call 00001562
[00001d23] 83c408          add esp,+08
[00001d26] 8945fc          mov [ebp-04],eax
[00001d29] 837dfc00        cmp dword [ebp-04],+00
[00001d2d] 7402            jz 00001d31
[00001d2f] ebfe            jmp 00001d2f
[00001d31] 8b45fc          mov eax,[ebp-04]
[00001d34] 8be5            mov esp,ebp
[00001d36] 5d              pop ebp
[00001d37] c3              ret
Size in bytes:(0038) [00001d37]

_main()
[00001d72] 55              push ebp
[00001d73] 8bec            mov ebp,esp
[00001d75] 68121d0000      push 00001d12
[00001d7a] e893ffffff      call 00001d12
[00001d7f] 83c404          add esp,+04
[00001d82] 50              push eax
[00001d83] 6883070000      push 00000783
[00001d88] e815eaffff      call 000007a2
[00001d8d] 83c408          add esp,+08
[00001d90] 33c0            xor eax,eax
[00001d92] 5d              pop ebp
[00001d93] c3              ret
Size in bytes:(0034) [00001d93]
```

```
 machine   stack     stack     machine    assembly
 address   address   data      code       language
 ========  ========  ========  =========  =============
[00001d72][0010304e][00000000] 55         push ebp
[00001d73][0010304e][00000000] 8bec       mov ebp,esp
[00001d75][0010304a][00001d12] 68121d0000 push 00001d12 ; push address of D
[00001d7a][00103046][00001d7f] e893ffffff call 00001d12 ; execute D
[00001d12][00103042][0010304e] 55         push ebp
[00001d13][00103042][0010304e] 8bec       mov ebp,esp
[00001d15][0010303e][00000000] 51         push ecx
[00001d16][0010303e][00000000] 8b4508     mov eax,[ebp+08]
[00001d19][0010303a][00001d12] 50         push eax
[00001d1a][0010303a][00001d12] 8b4d08     mov ecx,[ebp+08]
[00001d1d][00103036][00001d12] 51         push ecx
[00001d1e][00103032][00001d23] e83ff8ffff call 00001562
```

```
H: Begin Simulation   Execution Trace Stored at:1130fa
Address_of_H:1562
[00001d12][001130e6][001130ea] 55         push ebp       ; begin simulated D
[00001d13][001130e6][001130ea] 8bec       mov ebp,esp
[00001d15][001130e2][001030b6] 51         push ecx
[00001d16][001130e2][001030b6] 8b4508     mov eax,[ebp+08]
[00001d19][001130de][00001d12] 50         push eax       ; push address of D
[00001d1a][001130de][00001d12] 8b4d08     mov ecx,[ebp+08]
[00001d1d][001130da][00001d12] 51         push ecx       ; push address of D
[00001d1e][001130d6][00001d23] e83ff8ffff call 00001562  ; call H
H: Infinitely Recursive Simulation Detected Simulation Stopped
```

We can see that the first seven instructions of D simulated by H precisely match the first seven instructions of the x86 source-code of D. This conclusively proves that these instructions were simulated correctly.

Anyone sufficiently technically competent in the x86 programming language will agree that the above execution trace of D simulated by H shows that D will never stop running unless H aborts its simulation of D.

H detects that D is calling itself with the exact same arguments that H was called with and there are no conditional branch instructions from the beginning of D to its call to H that can possibly escape the repetition of this recursive simulation.

```
[00001d23][0010303e][00000000] 83c408     add esp,+08
[00001d26][0010303e][00000000] 8945fc     mov [ebp-04],eax
[00001d29][0010303e][00000000] 837dfc00   cmp dword [ebp-04],+00
[00001d2d][0010303e][00000000] 7402       jz 00001d31
[00001d31][0010303e][00000000] 8b45fc     mov eax,[ebp-04]
[00001d34][00103042][0010304e] 8be5       mov esp,ebp
[00001d36][00103046][00001d7f] 5d         pop ebp
[00001d37][0010304a][00001d12] c3         ret
[00001d7f][0010304e][00000000] 83c404     add esp,+04
[00001d82][0010304a][00000000] 50         push eax
[00001d83][00103046][00000783] 6883070000 push 00000783
[00001d88][00103046][00000783] e815eaffff call 000007a2
Input_Halts = 0
[00001d8d][0010304e][00000000] 83c408     add esp,+08
[00001d90][0010304e][00000000] 33c0       xor eax,eax
[00001d92][00103052][00000018] 5d         pop ebp
[00001d93][00103056][00000000] c3         ret
Number of Instructions Executed(990) == 15 Pages
```

When H1 correctly simulates D it finds that D halts

```
int D(int (*x)())
{
  int Halt_Status = H(x, x);
  if (Halt_Status)    // This code is never reached on D(D)
    HERE: goto HERE; // This code is never reached on D(D)
  return Halt_Status;
}

int main()
{
  Output("Input_Halts = ", H1(D,D));
}

_D()
[00001d12] 55              push ebp
[00001d13] 8bec            mov ebp,esp
[00001d15] 51              push ecx
[00001d16] 8b4508          mov eax,[ebp+08]
[00001d19] 50              push eax
[00001d1a] 8b4d08          mov ecx,[ebp+08]
[00001d1d] 51              push ecx
[00001d1e] e83ff8ffff      call 00001562
[00001d23] 83c408          add esp,+08
[00001d26] 8945fc          mov [ebp-04],eax
[00001d29] 837dfc00        cmp dword [ebp-04],+00
[00001d2d] 7402            jz 00001d31
[00001d2f] ebfe            jmp 00001d2f
[00001d31] 8b45fc          mov eax,[ebp-04]
[00001d34] 8be5            mov esp,ebp
[00001d36] 5d              pop ebp
[00001d37] c3              ret
Size in bytes:(0038) [00001d37]

_main()
[00001d72] 55              push ebp
[00001d73] 8bec            mov ebp,esp
[00001d75] 68121d0000      push 00001d12
[00001d7a] 68121d0000      push 00001d12
[00001d7f] e8def6ffff      call 00001462
[00001d84] 83c408          add esp,+08
[00001d87] 50              push eax
[00001d88] 6883070000      push 00000783
[00001d8d] e810eaffff      call 000007a2
[00001d92] 83c408          add esp,+08
[00001d95] 33c0            xor eax,eax
[00001d97] 5d              pop ebp
[00001d98] c3              ret
Size in bytes:(0039) [00001d98]

 machine   stack     stack     machine    assembly
 address   address   data      code       language
 ========  ========  ========  =========  =============
[00001d72][0010305d][00000000] 55         push ebp
[00001d73][0010305d][00000000] 8bec       mov ebp,esp
[00001d75][00103059][00001d12] 68121d0000 push 00001d12 ; push address of D
[00001d7a][00103055][00001d12] 68121d0000 push 00001d12 ; push address of D
[00001d7f][00103051][00001d84] e8def6ffff call 00001462 ; call H1
```

```
H1: Begin Simulation   Execution Trace Stored at:113109
Address_of_H1:1462
[00001d12][001130f5][001130f9] 55         push ebp       ; begin simulated D
[00001d13][001130f5][001130f9] 8bec       mov ebp,esp
[00001d15][001130f1][001030c5] 51         push ecx
[00001d16][001130f1][001030c5] 8b4508     mov eax,[ebp+08]
[00001d19][001130ed][00001d12] 50         push eax       ; push address of D
[00001d1a][001130ed][00001d12] 8b4d08     mov ecx,[ebp+08]
[00001d1d][001130e9][00001d12] 51         push ecx       ; push address of D
[00001d1e][001130e5][00001d23] e83ff8ffff call 00001562  ; call H

H: Begin Simulation   Execution Trace Stored at:15db31
Address_of_H:1562
[00001d12][0015db1d][0015db21] 55         push ebp       ; begin simulated D
[00001d13][0015db1d][0015db21] 8bec       mov ebp,esp
[00001d15][0015db19][0014daed] 51         push ecx
[00001d16][0015db19][0014daed] 8b4508     mov eax,[ebp+08]
[00001d19][0015db15][00001d12] 50         push eax       ; push address of D
[00001d1a][0015db15][00001d12] 8b4d08     mov ecx,[ebp+08]
[00001d1d][0015db11][00001d12] 51         push ecx       ; push address of D
[00001d1e][0015db0d][00001d23] e83ff8ffff call 00001562  ; call H
H: Infinitely Recursive Simulation Detected Simulation Stopped
```

We can see that the first seven instructions of D simulated by H precisely match the first seven instructions of the x86 source-code of D. This conclusively proves that these instructions were simulated correctly.

Anyone sufficiently technically competent in the x86 programming language will agree that the above execution trace of D simulated by H shows that D will never stop running unless H aborts its simulation of D.

H detects that D is calling itself with the exact same arguments that H was called with and there are no conditional branch instructions from the beginning of D to its call to H that can possibly escape the repetition of this recursive simulation.

```
[00001d23][001130f1][001030c5] 83c408     add esp,+08
[00001d26][001130f1][00000000] 8945fc     mov [ebp-04],eax
[00001d29][001130f1][00000000] 837dfc00   cmp dword [ebp-04],+00
[00001d2d][001130f1][00000000] 7402       jz 00001d31
[00001d31][001130f1][00000000] 8b45fc     mov eax,[ebp-04]
[00001d34][001130f5][001130f9] 8be5       mov esp,ebp
[00001d36][001130f9][00001561] 5d         pop ebp
[00001d37][001130fd][00001d12] c3         ret
H1: End Simulation   Input Terminated Normally

[00001d84][0010305d][00000000] 83c408     add esp,+08
[00001d87][00103059][00000001] 50         push eax
[00001d88][00103055][00000783] 6883070000 push 00000783
[00001d8d][00103055][00000783] e810eaffff call 000007a2
Input_Halts = 1
[00001d92][0010305d][00000000] 83c408     add esp,+08
[00001d95][0010305d][00000000] 33c0       xor eax,eax
[00001d97][00103061][00000018] 5d         pop ebp
[00001d98][00103065][00000000] c3         ret
Number of Instructions Executed(470247) == 7019 Pages
```

When HH correctly simulates DD it finds that DD remains stuck in recursive simulation

```
void PP(ptr x)
{
  int Halt_Status = HH(x, x);
  if (Halt_Status)
    HERE: goto HERE;
  return;
}

int main()
{
  Output("Input_Halts = ", HH(PP, PP));
}

_PP()
[00001be2] 55              push ebp
[00001be3] 8bec            mov ebp,esp
[00001be5] 51              push ecx
[00001be6] 8b4508          mov eax,[ebp+08]
[00001be9] 50              push eax
[00001bea] 8b4d08          mov ecx,[ebp+08]
[00001bed] 51              push ecx
[00001bee] e88ff7ffff      call 00001382
[00001bf3] 83c408          add esp,+08
[00001bf6] 8945fc          mov [ebp-04],eax
[00001bf9] 837dfc00        cmp dword [ebp-04],+00
[00001bfd] 7402            jz 00001c01
[00001bff] ebfe            jmp 00001bff
[00001c01] 8be5            mov esp,ebp
[00001c03] 5d              pop ebp
[00001c04] c3              ret
Size in bytes:(0035) [00001c04]

_main()
[00001d72] 55              push ebp
[00001d73] 8bec            mov ebp,esp
[00001d75] 68e21b0000      push 00001be2
[00001d7a] 68e21b0000      push 00001be2
[00001d7f] e8fef5ffff      call 00001382
[00001d84] 83c408          add esp,+08
[00001d87] 50              push eax
[00001d88] 6883070000      push 00000783
[00001d8d] e810eaffff      call 000007a2
[00001d92] 83c408          add esp,+08
[00001d95] 33c0            xor eax,eax
[00001d97] 5d              pop ebp
[00001d98] c3              ret
Size in bytes:(0039) [00001d98]

 machine   stack     stack     machine    assembly
 address   address   data      code       language
 ========  ========  ========  =========  =============
[00001d72][0010305d][00000000] 55         push ebp
[00001d73][0010305d][00000000] 8bec       mov ebp,esp
[00001d75][00103059][00001be2] 68e21b0000 push 00001be2
[00001d7a][00103055][00001be2] 68e21b0000 push 00001be2
[00001d7f][00103051][00001d84] e8fef5ffff call 00001382
New slave_stack at:103101
```

```
Begin Local Halt Decider Simulation   Execution Trace Stored at:113109
[00001be2][001130f5][001130f9] 55         push ebp      // begin PP
[00001be3][001130f5][001130f9] 8bec       mov ebp,esp
[00001be5][001130f1][001030c5] 51         push ecx
[00001be6][001130f1][001030c5] 8b4508     mov eax,[ebp+08]
[00001be9][001130ed][00001be2] 50         push eax      // push address of PP
[00001bea][001130ed][00001be2] 8b4d08     mov ecx,[ebp+08]
[00001bed][001130e9][00001be2] 51         push ecx      // push address of PP
[00001bee][001130e5][00001bf3] e88ff7ffff call 00001382 // call HH
New slave_stack at:14db29
[00001be2][0015db1d][0015db21] 55         push ebp      // begin PP
[00001be3][0015db1d][0015db21] 8bec       mov ebp,esp
[00001be5][0015db19][0014daed] 51         push ecx
[00001be6][0015db19][0014daed] 8b4508     mov eax,[ebp+08]
[00001be9][0015db15][00001be2] 50         push eax      // push address of PP
[00001bea][0015db15][00001be2] 8b4d08     mov ecx,[ebp+08]
[00001bed][0015db11][00001be2] 51         push ecx      // push address of PP
[00001bee][0015db0d][00001bf3] e88ff7ffff call 00001382 // call HH
Local Halt Decider: Infinite Recursion Detected Simulation Stopped
```

It is completely obvious that when HH(PP,PP) correctly emulates its input that it must emulate the first eight instructions of PP. Because the eighth instruction of PP repeats this process we can know with complete certainty that the emulated PP never reaches its final “ret” instruction, thus never halts.

```
[00001d84][0010305d][00000000] 83c408     add esp,+08
[00001d87][00103059][00000000] 50         push eax
[00001d88][00103055][00000783] 6883070000 push 00000783
[00001d8d][00103055][00000783] e810eaffff call 000007a2
Input_Halts = 0
[00001d92][0010305d][00000000] 83c408     add esp,+08
[00001d95][0010305d][00000000] 33c0       xor eax,eax
[00001d97][00103061][00000018] 5d         pop ebp
[00001d98][00103065][00000000] c3         ret
Number of Instructions Executed(16828) == 251 Pages
```