Halting problem proofs refuted on the basis of software engineering

This is an explanation of a key new insight into the halting problem provided in the language of software engineering. Technical computer science terms are explained using software engineering terms. No knowledge of the halting problem is required.

It is based on fully operational software executed in the x86utm operating system. The x86utm operating system (based on an excellent open source x86 emulator) was created to study the details of the halting problem proof counter-examples at the much higher level of abstraction of C/x86.

To fully understand this paper a software engineer must be an expert in:
(a) The C programming language.
(b) The x86 programming language.
(c) Exactly how C translates into x86 (how C function calls are implemented in x86).
(d) The ability to recognize infinite recursion at the x86 assembly language level.

The computer science term “halting” means that a Turing Machine terminated normally reaching its last instruction known as its “final state”. This is the same idea as when a function returns to its caller as opposed to and contrast with getting stuck in an infinite loop or infinite recursion.

> 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 P, called with some input, can pass its own source and its input to H and then specifically do the opposite of what H predicts P will do. **No H can exist that handles this case**.
> https://en.wikipedia.org/wiki/Halting_problem

The following H and P have the above specified pathological relationship to each other. When H(P,P) correctly determines that its input specifies a non-halting sequence of instructions the above template is refuted. All of the conventional halting problem proofs depend on the above undecidable input template and fail without it.

```
#include <stdint.h>  // To keep things simple a single data type is used:
#define u32 uint32_t // 32-bit unsigned integer is compiled in 32-bit mode

void P(u32 x)
{
  if (H(x, x))
    HERE: goto HERE;
  return;
}

int main()
{
  Output("Input_Halts = ", H((u32)P, (u32)P));
}
```

A halt decider must compute the mapping from its inputs to an accept or reject state on the basis of the actual behavior that is actually specified by these inputs.

This general principle refutes conventional halting problem proofs
Every simulating halt decider that correctly simulates its input until it correctly predicts that this simulated input would never reach its final state, correctly rejects this input as non-halting.

From a purely software engineering perspective H(P,P) is required to correctly predict that its correct and complete x86 emulation of its input would never reach the "ret" instruction of this input and H must do this in a finite number of steps. **(see Example 03)**.

Example 01: H0 correctly determines that Infinite_Loop() never halts

```
void Infinite_Loop()
{
  HERE: goto HERE;
}

int main()
{
  Output("Input_Halts = ", H0((u32)Infinite_Loop));
}

_Infinite_Loop()
[00001102](01)  55         push ebp
[00001103](02)  8bec       mov ebp,esp
[00001105](02)  ebfe       jmp 00001105
[00001107](01)  5d         pop ebp
[00001108](01)  c3         ret
Size in bytes:(0007) [00001108]

_main()
[00001192](01)  55         push ebp
[00001193](02)  8bec       mov ebp,esp
[00001195](05)  6802110000 push 00001102
[0000119a](05)  e8d3fbffff call 00000d72
[0000119f](03)  83c404     add esp,+04
[000011a2](01)  50         push eax
[000011a3](05)  68a3040000 push 000004a3
[000011a8](05)  e845f3ffff call 000004f2
[000011ad](03)  83c408     add esp,+08
[000011b0](02)  33c0       xor eax,eax
[000011b2](01)  5d         pop ebp
[000011b3](01)  c3         ret
Size in bytes:(0034) [000011b3]

 machine   stack     stack     machine    assembly
 address   address   data      code       language
 ========  ========  ========  =========  =============
[00001192][00101ef8][00000000] 55         push ebp
[00001193][00101ef8][00000000] 8bec       mov ebp,esp
[00001195][00101ef4][00001102] 6802110000 push 00001102
[0000119a][00101ef0][0000119f] e8d3fbffff call 00000d72

H0: Begin Simulation   Execution Trace Stored at:211fac
[00001102][00211f9c][00211fa0] 55         push ebp
[00001103][00211f9c][00211fa0] 8bec       mov ebp,esp
[00001105][00211f9c][00211fa0] ebfe       jmp 00001105
[00001105][00211f9c][00211fa0] ebfe       jmp 00001105
H0: Infinite Loop Detected Simulation Stopped

 if (current->Simplified_Opcode == JMP)           // JMP
  if (current->Decode_Target <= current->Address) // upward
   if (traced->Address == current->Decode_Target) // to this address
    if (Conditional_Branch_Count == 0)            // no escape
     return 1;

[0000119f][00101ef8][00000000] 83c404     add esp,+04
[000011a2][00101ef4][00000000] 50         push eax
[000011a3][00101ef0][000004a3] 68a3040000 push 000004a3
[000011a8][00101ef0][000004a3] e845f3ffff call 000004f2
Input_Halts = 0
[000011ad][00101ef8][00000000] 83c408     add esp,+08
[000011b0][00101ef8][00000000] 33c0       xor eax,eax
[000011b2][00101efc][00100000] 5d         pop ebp
[000011b3][00101f00][00000004] c3         ret
Number of Instructions Executed(554) == 8 Pages
```

Example 02: H correctly determines that Infinite_Recursion() never halts

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

int main()
{
  Output("Input_Halts = ", H((u32)Infinite_Recursion, 0x777));
}

_Infinite_Recursion()
[000010f2](01)  55         push ebp
[000010f3](02)  8bec       mov ebp,esp
[000010f5](03)  8b4508     mov eax,[ebp+08]
[000010f8](01)  50         push eax
[000010f9](05)  e8f4ffffff call 000010f2
[000010fe](03)  83c404     add esp,+04
[00001101](01)  5d         pop ebp
[00001102](01)  c3         ret
Size in bytes:(0017) [00001102]

_main()
[000011b2](01)  55         push ebp
[000011b3](02)  8bec       mov ebp,esp
[000011b5](05)  6877070000 push 00000777
[000011ba](05)  68f2100000 push 000010f2
[000011bf](05)  e8aefdffff call 00000f72
[000011c4](03)  83c408     add esp,+08
[000011c7](01)  50         push eax
[000011c8](05)  68a3040000 push 000004a3
[000011cd](05)  e820f3ffff call 000004f2
[000011d2](03)  83c408     add esp,+08
[000011d5](02)  33c0       xor eax,eax
[000011d7](01)  5d         pop ebp
[000011d8](01)  c3         ret
Size in bytes:(0039) [000011d8]

 machine   stack     stack     machine    assembly
 address   address   data      code       language
 ========  ========  ========  =========  =============
[000011b2][00101f39][00000000] 55         push ebp
[000011b3][00101f39][00000000] 8bec       mov ebp,esp
[000011b5][00101f35][00000777] 6877070000 push 00000777
[000011ba][00101f31][000010f2] 68f2100000 push 000010f2
[000011bf][00101f2d][000011c4] e8aefdffff call 00000f72

H: Begin Simulation   Execution Trace Stored at:111fe5
[000010f2][00111fd1][00111fd5] 55         push ebp
[000010f3][00111fd1][00111fd5] 8bec       mov ebp,esp
[000010f5][00111fd1][00111fd5] 8b4508     mov eax,[ebp+08]
[000010f8][00111fcd][00000777] 50         push eax      // push 0x777
[000010f9][00111fc9][000010fe] e8f4ffffff call 000010f2 // call Infinite_Recursion
[000010f2][00111fc5][00111fd1] 55         push ebp
[000010f3][00111fc5][00111fd1] 8bec       mov ebp,esp
[000010f5][00111fc5][00111fd1] 8b4508     mov eax,[ebp+08]
[000010f8][00111fc1][00000777] 50         push eax      // push 0x777
[000010f9][00111fbd][000010fe] e8f4ffffff call 000010f2 // call Infinite_Recursion
H: Infinite Recursion Detected Simulation Stopped

 if (current->Simplified_Opcode == CALL)
  if (current->Simplified_Opcode == traced->Simplified_Opcode) // CALL
   if (current->Address == traced->Address)                  // from same address
    if (current->Decode_Target == traced->Decode_Target)      // to Same Function
     if (Conditional_Branch_Count == 0)                       // no escape
      return 2;

[000011c4][00101f39][00000000] 83c408     add esp,+08
[000011c7][00101f35][00000000] 50         push eax
[000011c8][00101f31][000004a3] 68a3040000 push 000004a3
[000011cd][00101f31][000004a3] e820f3ffff call 000004f2
Input_Halts = 0
[000011d2][00101f39][00000000] 83c408     add esp,+08
[000011d5][00101f39][00000000] 33c0       xor eax,eax
[000011d7][00101f3d][00000018] 5d         pop ebp
[000011d8][00101f41][00000000] c3         ret
Number of Instructions Executed(1118) == 17 Pages
```

Example 03: H(P,P) correctly determines that its input never halts

```
void P(u32 x)
{
  if (H(x, x))
    HERE: goto HERE;
  return;
}

int main()
{
  Output("Input_Halts = ", H((u32)P, (u32)P));
}

_P()
[00001202](01)  55              push ebp
[00001203](02)  8bec            mov ebp,esp
[00001205](03)  8b4508          mov eax,[ebp+08]
[00001208](01)  50              push eax
[00001209](03)  8b4d08          mov ecx,[ebp+08]
[0000120c](01)  51              push ecx
[0000120d](05)  e820feffff      call 00001032
[00001212](03)  83c408          add esp,+08
[00001215](02)  85c0            test eax,eax
[00001217](02)  7402            jz 0000121b
[00001219](02)  ebfe            jmp 00001219
[0000121b](01)  5d              pop ebp
[0000121c](01)  c3              ret
Size in bytes:(0027) [0000121c]

_main()
[00001222](01)  55              push ebp
[00001223](02)  8bec            mov ebp,esp
[00001225](05)  6802120000      push 00001202
[0000122a](05)  6802120000      push 00001202
[0000122f](05)  e8fefdffff      call 00001032
[00001234](03)  83c408          add esp,+08
[00001237](01)  50              push eax
[00001238](05)  68b3030000      push 000003b3
[0000123d](05)  e8c0f1ffff      call 00000402
[00001242](03)  83c408          add esp,+08
[00001245](02)  33c0            xor eax,eax
[00001247](01)  5d              pop ebp
[00001248](01)  c3              ret
Size in bytes:(0039) [00001248]

 machine   stack     stack     machine    assembly
 address   address   data      code       language
 ========  ========  ========  =========  =============
[00001222][0010200f][00000000] 55         push ebp
[00001223][0010200f][00000000] 8bec       mov ebp,esp
[00001225][0010200b][00001202] 6802120000 push 00001202 // push P
[0000122a][00102007][00001202] 6802120000 push 00001202 // push P
[0000122f][00102003][00001234] e8fefdffff call 00001032 // call executed H

Begin Simulation   Execution Trace Stored at:2120c3
Address_of_H:1032
[00001202][002120af][002120b3] 55         push ebp
[00001203][002120af][002120b3] 8bec       mov ebp,esp
[00001205][002120af][002120b3] 8b4508     mov eax,[ebp+08]
[00001208][002120ab][00001202] 50         push eax      // push P
[00001209][002120ab][00001202] 8b4d08     mov ecx,[ebp+08]
[0000120c][002120a7][00001202] 51         push ecx      // push P
[0000120d][002120a3][00001212] e820feffff call 00001032 // call emulated H
Infinitely Recursive Simulation Detected Simulation Stopped
```

H knows its own machine address and on this basis it can easily examine its stored execution_trace of P (see above) to determine:
(a) P is calling H with the same arguments that H was called with.
(b) No instructions in P could possibly escape this otherwise infinitely recursive emulation.
(c) H aborts its emulation of P before its call to H is emulated.

```
[00001234][0010200f][00000000] 83c408     add esp,+08
[00001237][0010200b][00000000] 50         push eax
[00001238][00102007][000003b3] 68b3030000 push 000003b3
[0000123d][00102007][000003b3] e8c0f1ffff call 00000402
Input_Halts = 0
[00001242][0010200f][00000000] 83c408     add esp,+08
[00001245][0010200f][00000000] 33c0       xor eax,eax
[00001247][00102013][00100000] 5d         pop ebp
[00001248][00102017][00000004] c3         ret
Number of Instructions Executed(870) / 67 = 13 pages
```

From a purely software engineering perspective (anchored in the semantics of the x86 language) it is proven that H(P,P) correctly predicts that its correct and complete x86 emulation of its input would never reach the "ret" instruction (final state) of this input.

Example 04: An impossible program: Strachey(1965)

The Computer Journal, Volume 7, Issue 4, January 1965, Page 313,
https://doi.org/10.1093/comjnl/7.4.313

```
typedef void (*ptr)();
// rec routine P
//   §L :if T[P] go to L
//     Return §
void Strachey_P()
{
  L: if (T(Strachey_P)) goto L;
  return;
}

int main()
{
  Output("Input_Halts = ", T(Strachey_P));
}

_Strachey_P()
[000012a6](01)  55         push ebp
[000012a7](02)  8bec       mov ebp,esp
[000012a9](05)  68a6120000 push 000012a6
[000012ae](05)  e833fcffff call 00000ee6
[000012b3](03)  83c404     add esp,+04
[000012b6](02)  85c0       test eax,eax
[000012b8](02)  7402       jz 000012bc
[000012ba](02)  ebed       jmp 000012a9
[000012bc](01)  5d         pop ebp
[000012bd](01)  c3         ret
Size in bytes:(0024) [000012bd]

_main()
[00001346](01)  55         push ebp
[00001347](02)  8bec       mov ebp,esp
[00001349](05)  68a6120000 push 000012a6
[0000134e](05)  e893fbffff call 00000ee6
[00001353](03)  83c404     add esp,+04
[00001356](01)  50         push eax
[00001357](05)  6817050000 push 00000517
[0000135c](05)  e805f2ffff call 00000566
[00001361](03)  83c408     add esp,+08
[00001364](02)  33c0       xor eax,eax
[00001366](01)  5d         pop ebp
[00001367](01)  c3         ret
Size in bytes:(0034) [00001367]

 machine   stack     stack     machine    assembly
 address   address   data      code       language
 ========  ========  ========  =========  =============
[00001346][0010221b][00000000] 55         push ebp
[00001347][0010221b][00000000] 8bec       mov ebp,esp
[00001349][00102217][000012a6] 68a6120000 push 000012a6
[0000134e][00102213][00001353] e893fbffff call 00000ee6

T: Begin Simulation   Execution Trace Stored at:1122c7
Address_of_T:ee6
[000012a6][001122b7][001122bb] 55         push ebp
[000012a7][001122b7][001122bb] 8bec       mov ebp,esp
[000012a9][001122b3][000012a6] 68a6120000 push 000012a6
[000012ae][001122af][000012b3] e833fcffff call 00000ee6
T: Infinitely Recursive Simulation Detected Simulation Stopped
```

T knows its own machine address and on this basis it can easily examine its stored execution_trace of Strachey_P (see above) to determine:
(a) Strachey_P is calling T with the same arguments that T was called with.
(b) No instructions in Strachey_P could possibly escape this otherwise infinitely recursive emulation.
(c) T aborts its emulation of Strachey_P before its call to T is emulated.

```
[00001353][0010221b][00000000] 83c404     add esp,+04
[00001356][00102217][00000000] 50         push eax
[00001357][00102213][00000517] 6817050000 push 00000517
[0000135c][00102213][00000517] e805f2ffff call 00000566
Input_Halts = 0
[00001361][0010221b][00000000] 83c408     add esp,+08
[00001364][0010221b][00000000] 33c0       xor eax,eax
[00001366][0010221f][00000018] 5d         pop ebp
[00001367][00102223][00000000] c3         ret
Number of Instructions Executed(538) == 8 Pages
```

Appendix (Simulating halt decider applied to Peter Linz proof)

The following is the same idea a shown above this time it is applied to the Peter Linz Halting Problem proof. It can only be undertood within the context of this proof.

A simulating halt decider (SHD) computes the mapping from its inputs to its own final states on the basis of the behavior of its correctly simulated input.

All of the conventional halting problem counter-example inputs are simply rejected by a simulating halt decider as non-halting because they fail to meet the Linz definition of halting:

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

USENET comp.theory: On 4/11/2022 3:19 PM, Malcolm McLean wrote:
> PO's idea is to have a simulator with an infinite cycle detector.
> You would achieve this by modifying a UTM, so describing it as
> a "modified UTM", or "acts like a UTM until it detects an infinite
> cycle", is reasonable. And such a machine is a fairly powerful
> halt decider. Even if the infinite cycle detector isn't very
> sophisticated, it will still catch a large subset of non-halting
> machines.

The following simplifies the syntax for the definition of the Linz Turing machine Ĥ.
There is no need for the infinite loop after H.qy because it is never reached. The halting criteria has been adapted so that it applies to a simulating halt decider (SHD).

$\hat{H}.q_0 \langle \hat{H} \rangle \vdash^* H \langle \hat{H} \rangle \langle \hat{H} \rangle \vdash^* \hat{H}.qy$
If the correctly simulated input ⟨Ĥ⟩ ⟨Ĥ⟩ to H would reach its own final state of ⟨Ĥ.qy⟩ or ⟨Ĥ.qn⟩.

$\hat{H}.q_0 \langle \hat{H} \rangle \vdash^* H \langle \hat{H} \rangle \langle \hat{H} \rangle \vdash^* \hat{H}.qn$
If the correctly simulated input ⟨Ĥ⟩ ⟨Ĥ⟩ to H would never reach its own final state of ⟨Ĥ.qy⟩ or ⟨Ĥ.qn⟩.

When Ĥ is applied to ⟨Ĥ⟩ // subscripts indicate unique finite strings
Ĥ copies its input $\langle \hat{H}_0 \rangle$ to $\langle \hat{H}_1 \rangle$ then H simulates $\langle \hat{H}_0 \rangle \langle \hat{H}_1 \rangle$

Then these steps would keep repeating: (unless their simulation is aborted)
$\hat{H}_0$ copies its input $\langle \hat{H}_1 \rangle$ to $\langle \hat{H}_2 \rangle$ then H_0 simulates $\langle \hat{H}_1 \rangle \langle \hat{H}_2 \rangle$
$\hat{H}_1$ copies its input $\langle \hat{H}_2 \rangle$ to $\langle \hat{H}_3 \rangle$ then H_1 simulates $\langle \hat{H}_2 \rangle \langle \hat{H}_3 \rangle$
$\hat{H}_2$ copies its input $\langle \hat{H}_3 \rangle$ to $\langle \hat{H}_4 \rangle$ then H_2 simulates $\langle \hat{H}_3 \rangle \langle \hat{H}_4 \rangle$...

Since we can see that the simulated input: $\langle \hat{H}_0 \rangle$ to H would never reach its own final state of $\langle \hat{H}_0.qy \rangle$ or $\langle \hat{H}_0.qn \rangle$ we know that it is non-halting.

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