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](https://en.wikipedia.org/wiki/Halting_problem)

H and P implement the above specified pathological relationship to each other:

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

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

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 appendix).
Appendix (three examples)

H0 correctly determines that Infinite_Loop() never halts

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

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

Infinite_Loop()

```assembly
machine   stack     stack     machine    assembly
address   address   data      code       language
========  ========  ========  =========  =============
[00001102](01) 55  push ebp
[00001109](02) 8bec mov ebp,esp
[000011105](02) ebfe jmp 00001105
[00001107](01) 5d  pop ebp
[00001108](01) c3 ret
Size in bytes:(0007) [00001108]
```

main()

```assembly
[00001192](01) 55  push ebp
[00001193](02) 8bec mov ebp,esp
[00001195](05) 6802110000 push 00001102
[00001199](05) e8d3fbffff call 00000d72
[000011a4](03) 83c404 add esp,+04
[000011b0](02) 50  push eax
[000011b1](05) 68a3040000 push 000004a3
[000011b6](05) e845f3ffff call 000004f2
[000011b9](03) 83c408 add esp,+08
[000011ba](01) 5d  pop ebp
[000011bb](01) c3 ret
Size in bytes:(0034) [000011bb]
```

H0: Begin Simulation Execution Trace Stored at:211fac
```
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;
```

Input_Halts = 0
```
Number of Instructions Executed(554) == 8 Pages
H correctly determines that Infinite_Recursion() never halts

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

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

```assembly
_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)  83c404     add esp,+04
[000011c8] (05)  68a3040000 push 000004a3
[000011cd] (05)  6820f3ffff call 000004f2
[000011d2] (03)  83c404     add esp,+04
[000011d5] (02)  33c0       xor eax,eax
[000011d7] (01)  5d         pop ebp
[000011d8] (01)  c3         ret
Size in bytes: (0039) [000011d8]
```

```
---3--- 07/02/22 03:17:55 PM
```
H correctly determines that \texttt{P()} never halts

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

int \texttt{main()}
{ Output("Input Halts = ", H((u32)P, (u32)P));
}
```

```
\begin{verbatim}
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)  84d08           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]
\end{verbatim}
```

```
\begin{verbatim}
00001222(01)  55              push ebp
00001223(02)  8bec            mov ebp,esp
00001225(05)  6802120000      push 00001202
0000122a(05)  6802120000      push 00001202
0000122f(05)  e8fedffff       call 00001032
00001234(05)  83c408          add esp,+08
00001237(01)  50              push eax
00001238(05)  68b3030000      push 000003b3
0000123d(05)  e8c0f1ffff      call 00000402
00001242(05)  83c408          add esp,+08
00001245(02)  33c0            xor eax,eax
00001247(01)  5d              pop ebp
00001248(01)  c3              ret
Size in bytes:(0039) [00001248]
\end{verbatim}
```

```
\begin{verbatim}
Begin Simulation   Execution Trace Stored at:2120c3
Address_of_H:1032
\end{verbatim}

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

```
\begin{verbatim}
00001234(01)  0010200f [00000000] 83c408 add esp,+08
00001237(01)  0010200f [00000000] 50  push eax
00001238(01)  0010200f [000003b3] 68b303000 push 000003b3
0000123d(01)  0010200f [000003b3] e8c0f1ffff call 00000402
Input Halts = 0
\end{verbatim}
```

```
Copyright 2022 PL Olcott
---4---
 07/02/22 03:17:55 PM
```