## Eliminating pathological self-reference error from the halting problem

When-so-ever a question is defined such that both yes and no answers are contradicted we have a contradictory (thus erroneous) question.

When-so-ever a decision problem input is defined such that this input contradicts whatever decision that the decider makes we have a contradictory (thus erroneous) input to this decision problem.

If we ask Jack[1] this question: Will your answer to this question be no?
Neither answer of yes or no is correct because they are both contradicted.

When an input program is defined to do opposite of whatever the halt decider decides about its halt status then: Will the input program halt on its input? Becomes a contradictory question for this input.

Other people besides Jack can correctly answer whether or not Jack's answer will be no:
(a) If Jack says no and Bill says Jack will say no, Jack is wrong and Bill is right.
(b) If Jack says yes and Bill says Jack will say Yes, Jack is wrong and Bill is right.

For certain halt decider / input pairs the question:
**Will the input program halt on its input?**
Is a contradictory question.

This same contradictory question can be redefined eliminating its contradiction:
**Would the input program halt if the halt decider never stopped simulating it?**

This corrected criteria allows the set of inputs to be divided into those that would halt and those that would not.

The simplest way to define a halt decider is to make a program that runs its input to see what it does. In technical terms this would be a universal Turing machine (UTM) that has been adapted to become a halt decider. This UTM would simply simulate the execution of its input until its input halts on its own or the halt decider determines that its input would never halt on its own and stops simulating it. In order for the UTM to see what its input does it must keep track of an execution trace of its input.

Every (at least partial) halt decider that decides the halting status of its input on the basis of its examination of the execution trace of its own simulation of this input would correctly decide the conventional halting problem counter-examples would not halt.

The x86utm operating system was created so that halt deciders written in the C programming language would be computationally equivalent to the execution of actual Turing machines. These (at least partial) halt deciders base their halting decision on examining the execution trace of the x86 machine language of their input. The input is the COFF object file output of a C compiler and is directly executed by the x86 emulator.

The halt decider Halts() determines that H_Hat() called it in infinite recursion, on the basis of the x86 machine language execution trace shown below, stop simulating its input, and decide not halting on the basis that its input would not halt.

```
void H_Hat(u32 P)
{
  u32 Input_Would_Halt = Halts(P, P);
  if (Input_Would_Halt)
    HERE: goto HERE;
}


int main()
{
  u32 Input_Would_Halt = Halts((u32)H_Hat, (u32)H_Hat);
  Output("Input_Would_Halt = ", Input_Would_Halt);
}


_H_Hat()
[00000a63](01)  55              push ebp
[00000a64](02)  8bec            mov ebp,esp
[00000a66](01)  51              push ecx
[00000a67](03)  8b4508          mov eax,[ebp+08]
[00000a6a](01)  50              push eax
[00000a6b](03)  8b4d08          mov ecx,[ebp+08]
[00000a6e](01)  51              push ecx
[00000a6f](05)  e80ffeffff      call 00000883
[00000a74](03)  83c408          add esp,+08
[00000a77](03)  8945fc          mov [ebp-04],eax
[00000a7a](04)  837dfc00        cmp dword [ebp-04],+00
[00000a7e](02)  7402            jz 00000a82
[00000a80](02)  ebfe            jmp 00000a80
[00000a82](02)  8be5            mov esp,ebp
[00000a84](01)  5d              pop ebp
[00000a85](01)  c3              ret
Size in bytes:(0035)


_main()
[00000a93](01)  55              push ebp
[00000a94](02)  8bec            mov ebp,esp
[00000a96](01)  51              push ecx
[00000a97](05)  68630a0000      push 00000a63
[00000a9c](05)  68630a0000      push 00000a63
[00000aa1](05)  e8ddfdffff      call 00000883
[00000aa6](03)  83c408          add esp,+08
[00000aa9](03)  8945fc          mov [ebp-04],eax
[00000aac](03)  8b45fc          mov eax,[ebp-04]
[00000aaf](01)  50              push eax
[00000ab0](05)  681f030000      push 0000031f
[00000ab5](05)  e8a9f8ffff      call 00000363
[00000aba](03)  83c408          add esp,+08
[00000abd](02)  33c0            xor eax,eax
[00000abf](02)  8be5            mov esp,ebp
[00000ac1](01)  5d              pop ebp
[00000ac2](01)  c3              ret
Size in bytes:(0048)
```

**Columns**
(1) Execution trace sequence number
(2) Machine address of instruction
(3) Machine address of top of stack
(4) Value of top of stack after instruction executed
(5) Number of bytes of machine code
(6) Machine language bytes
(7) Assembly language text

```
(01)[00000a93][000114f7][00000000](01)  55              push ebp
(02)[00000a94][000114f7][00000000](02)  8bec            mov ebp,esp
(03)[00000a96][000114f3][00000000](01)  51              push ecx
(04)[00000a97][000114ef][00000a63](05)  68630a0000      push 00000a63
(05)[00000a9c][000114eb][00000a63](05)  68630a0000      push 00000a63
(06)[00000aa1][000114e7][00000aa6](05)  e8ddfdffff      call 00000883
(07)[00000a63][00031577][0003157b](01)  55              push ebp
(08)[00000a64][00031577][0003157b](02)  8bec            mov ebp,esp
(09)[00000a66][00031573][00021547](01)  51              push ecx
(10)[00000a67][00031573][00021547](03)  8b4508          mov eax,[ebp+08]
(11)[00000a6a][0003156f][00000a63](01)  50              push eax
(12)[00000a6b][0003156f][00000a63](03)  8b4d08          mov ecx,[ebp+08]
(13)[00000a6e][0003156b][00000a63](01)  51              push ecx
(14)[00000a6f][00031567][00000a74](05)  e80ffeffff      call 00000883
(15)[00000a63][0004271f][00042723](01)  55              push ebp
(16)[00000a64][0004271f][00042723](02)  8bec            mov ebp,esp
(17)[00000a66][0004271b][000326ef](01)  51              push ecx
(18)[00000a67][0004271b][000326ef](03)  8b4508          mov eax,[ebp+08]
(19)[00000a6a][00042717][00000a63](01)  50              push eax
(20)[00000a6b][00042717][00000a63](03)  8b4d08          mov ecx,[ebp+08]
(21)[00000a6e][00042713][00000a63](01)  51              push ecx
(22)[00000a6f][0004270f][00000a74](05)  e80ffeffff      call 00000883
Infinite Recursion Detected Simulation Stopped
```

Halts() detected that is was called twice in the execution trace of H_Hat() from the same machine address without any conditional branch instructions inbetween.

```
(23)[00000aa6][000114f3][00000000](03)  83c408          add esp,+08
(24)[00000aa9][000114f3][00000000](03)  8945fc          mov [ebp-04],eax
(25)[00000aac][000114f3][00000000](03)  8b45fc          mov eax,[ebp-04]
(26)[00000aaf][000114ef][00000000](01)  50              push eax
(27)[00000ab0][000114eb][0000031f](05)  681f030000      push 0000031f
(28)[00000ab5][000114e7][00000aba](05)  e8a9f8ffff      call 00000363
Input_Would_Halt = 0
(29)[00000aba][000114f3][00000000](03)  83c408          add esp,+08
(30)[00000abd][000114f3][00000000](02)  33c0            xor eax,eax
(31)[00000abf][000114f7][00000000](02)  8be5            mov esp,ebp
(32)[00000ac1][000114fb][00010000](01)  5d              pop ebp
(33)[00000ac2][000114ff][00000080](01)  c3              ret
Number_of_User_Instructions(33)
Number of Instructions Executed(13966)
```



[1]sci.logic --- Daryl McCullough --- Jun 25, 2004, 6:30:39 PM
[The Psychology of Self-Reference]
https://groups.google.com/g/sci.logic/c/4kIXI1kxmsI/m/hRroMoQZx2IJ