

## Halting problem undecidability and infinitely nested simulation

The x86utm operating system was created so that the halting problem could be examined concretely in the high level language of C. H is a function written in C that analyzes the x86 machine language execution trace of other functions written in C. H recognizes simple cases of infinite recursion and infinite loops. The conventional halting problem proof counter-example template is shown to simply be an input that does not halt.

H simulates its input with an x86 emulator until it determines that its input would never halt. As soon as H recognizes that its input would never halt it stops simulating this input and returns 0. For inputs that do halt H acts exactly as if it was an x86 emulator and simply runs its input to completion and then returns 1.

the Turing machine halting problem. Simply stated, the problem is: given the description of a Turing machine  $M$  and an input  $w$ , does  $M$ , when started in the initial configuration  $q_0w$ , perform a computation that eventually halts? (Linz:1990:317).

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. [https://en.wikipedia.org/wiki/Halting\\_problem](https://en.wikipedia.org/wiki/Halting_problem)

Because H only acts as a pure simulator of its input until after its halt status decision has been made it has no behavior that can possibly effect the behavior of its input. Because of this H screens out its own address range in every execution trace that it examines. This is why we never see any instructions of H in any execution trace after an input calls H.

**Halting computation:** is any computation that eventually reaches its own final state. This criteria divides computations that halt from those that merely stop running because their simulation was aborted.

A Turing machine is said to halt whenever it reaches a configuration for which  $\delta$  is not defined; this is possible because  $\delta$  is a partial function. In fact, we will assume that no transitions are defined for any final state, so the Turing machine will halt whenever it enters a final state. (Linz:1990:234)

**Pathological Input** to a halt decider is stipulated to mean any input that was defined to do the opposite of whatever its corresponding halt decider decides as Sipser describes:

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  $\langle M \rangle$ . Once  $D$  has determined this information, it does the opposite. (Sipser:1997:165)

## Does D halts on its own machine description (D) ?

This question can only be correctly answered after the pathology has been removed. When a halt decider only acts as a pure simulator of its input until after its halt status decision is made there is no feedback loop of back channel communication between the halt decider and its input that can prevent a correct halt status decision. In this case the halt decider is only examining the behavior of the input. It ignores its own behavior.

The standard pseudo-code halting problem template "proved" that the halting problem could never be solved on the basis that neither value of true (halting) nor false (not halting) could be correctly returned from the halt decider to the confounding input.

```
procedure compute_g(i): // (wikipedia:Halting Problem)
  if f(i, i) == 0 then // adapted from (Strachey, C 1965)
    return 0 // originally written in CPL
  else // ancestor of the BCPL, B and C
    loop forever // programming languages
```

This problem is overcome on the basis that a simulating halt decider would abort the simulation of its input before ever returning any value to this input. It aborts the simulation of its input on the basis that its input specifies what is essentially infinite recursion (infinitely nested simulation) to any simulating halt decider.

Every input to a simulating halt decider that only stops running when its simulation is aborted unequivocally specifies a computation that never halts. When input to a simulating halt decider cannot possibly reach its final state then we know that this input never halts.

## Simulating partial halt decider H correctly decides that P(P) never halts (V1)

H analyzes the (currently updated) stored execution trace of its x86 emulation of P(P) after it simulates each instruction of input (P, P). As soon as a non-halting behavior pattern is matched H aborts the simulation of its input and decides that its input never reaches its final state.

The execution trace of the x86 emulation of P(P) by simulating halt decider H conclusively proves that P cannot possibly ever reach its final state of 0xc50. This provides complete proof that the input to H never halts thus  $H(P,P)=0$  is correct.

```
// Simplified Linz A (Linz:1990:319)
// Strachey(1965) CPL translated to C
void P(u32 x)
```

```
{
    if (H(x, x))
        HERE: goto HERE;
}
```

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

```
_P()
[0000c36] (01) 55          push ebp
[0000c37] (02) 8bec         mov ebp,esp
[0000c39] (03) 8b4508      mov eax,[ebp+08] // 2nd Param
[0000c3c] (01) 50          push eax
[0000c3d] (03) 8b4d08      mov ecx,[ebp+08] // 1st Param
[0000c40] (01) 51          push ecx
[0000c41] (05) e820fdffff  call 00000966 // call H(P,P)
[0000c46] (03) 83c408      add esp,+08
[0000c49] (02) 85c0        test eax,eax
[0000c4b] (02) 7402        jz 0000c4f
[0000c4d] (02) ebfe        jmp 0000c4d
[0000c4f] (01) 5d          pop ebp
[0000c50] (01) c3          ret
Size in bytes:(0027) [0000c50]
```

```
_main()
[0000c56] (01) 55          push ebp
[0000c57] (02) 8bec         mov ebp,esp
[0000c59] (05) 68360c0000 push 0000c36 // push P
[0000c5e] (05) 68360c0000 push 0000c36 // push P
[0000c63] (05) e8fefcffff  call 00000966 // call H(P,P)
[0000c68] (03) 83c408      add esp,+08
[0000c6b] (01) 50          push eax
[0000c6c] (05) 6857030000 push 00000357
[0000c71] (05) e810f7ffff  call 00000386
[0000c76] (03) 83c408      add esp,+08
[0000c79] (02) 33c0        xor eax,eax
[0000c7b] (01) 5d          pop ebp
[0000c7c] (01) c3          ret
Size in bytes:(0039) [0000c7c]
```

machine address	stack address	stack data	machine code	assembly language
[0000c56]	[0010172a]	[00000000]	55	push ebp
[0000c57]	[0010172a]	[00000000]	8bec	mov ebp,esp
[0000c59]	[00101726]	[00000c36]	68360c0000	push 0000c36 // push P
[0000c5e]	[00101722]	[00000c36]	68360c0000	push 0000c36 // push P
[0000c63]	[0010171e]	[00000c68]	e8fefcffff	call 00000966 // call H(P,P)

**Begin Local Halt Decider Simulation at Machine Address:c36**

```
[0000c36] [002117ca] [002117ce] 55      push ebp
[0000c37] [002117ca] [002117ce] 8bec   mov ebp,esp
[0000c39] [002117ca] [002117ce] 8b4508 mov eax,[ebp+08]
[0000c3c] [002117c6] [0000c36] 50     push eax // push P
[0000c3d] [002117c6] [0000c36] 8b4d08 mov ecx,[ebp+08]
[0000c40] [002117c2] [0000c36] 51     push ecx // push P
[0000c41] [002117be] [0000c46] e820fdffff call 00000966 // call H(P,P)
```

```
[0000c36] [0025c1f2] [0025c1f6] 55      push ebp
[0000c37] [0025c1f2] [0025c1f6] 8bec   mov ebp,esp
[0000c39] [0025c1f2] [0025c1f6] 8b4508 mov eax,[ebp+08]
[0000c3c] [0025c1ee] [0000c36] 50     push eax // push P
[0000c3d] [0025c1ee] [0000c36] 8b4d08 mov ecx,[ebp+08]
[0000c40] [0025c1ea] [0000c36] 51     push ecx // push P
[0000c41] [0025c1e6] [0000c46] e820fdffff call 00000966 // call H(P,P)
```

**Local Halt Decider: Infinite Recursion Detected Simulation Stopped**

In the above 14 instructions of the simulation of P(P) we can see that the first 7 instructions of P are repeated. The end of this sequence of 7 instructions P calls H with its own machine address as the parameters to H(P,P). Because H only examines the behavior of its inputs and ignores its own behavior when H(P,P) is called we only see the first instruction of P being simulated.

Anyone knowing the x86 language well enough can see that none of these 7 simulated instructions of P have any escape from their infinitely repeating behavior pattern. When H recognizes this infinitely repeating pattern it aborts its simulation of P(P) and reports that its input: (P,P) never reaches its final state of 0xc50.

```
[0000c68] [0010172a] [00000000] 83c408 add esp,+08
[0000c6b] [00101726] [00000000] 50     push eax
[0000c6c] [00101722] [00000357] 6857030000 push 00000357
[0000c71] [00101722] [00000357] e810f7ffff call 00000386
Input_Halts = 0
[0000c76] [0010172a] [00000000] 83c408 add esp,+08
[0000c79] [0010172a] [00000000] 33c0   xor eax,eax
[0000c7b] [0010172e] [00100000] 5d     pop ebp
[0000c7c] [00101732] [00000068] c3     ret
Number_of_User_Instructions(27)
Number of Instructions Executed(23721)
```

## Simulating partial halt decider H correctly decides that P(P) never halts (V2)

```

int simulate(u32 P, u32 I)
{
    ((int(*) (int))P)(I);
    return 1;
}

// Simplified Linz A (Linz:1990:319)
// Strachey(1965) CPL translated to C
void P(u32 x)
{
    if (H(x, x))
        HERE: goto HERE;
}

int main()
{
    output("Input_Halts = ", simulate((u32)P, (u32)P));
}

```

```

_simulate()
[00000ce2] (01) 55          push ebp
[00000ce3] (02) 8bec          mov ebp,esp
[00000ce5] (03) 8b450c        mov eax,[ebp+0c]
[00000ce8] (01) 50          push eax // push 2nd Param
[00000ce9] (03) ff5508        call dword [ebp+08] // call 1st Param
[00000cec] (03) 83c404        add esp,+04
[00000cef] (05) b801000000    mov eax,00000001
[00000cf4] (01) 5d          pop ebp
[00000cf5] (01) c3          ret
Size in bytes:(0020) [00000cf5]

```

```

_P()
[00000d02] (01) 55          push ebp
[00000d03] (02) 8bec          mov ebp,esp
[00000d05] (03) 8b4508        mov eax,[ebp+08]
[00000d08] (01) 50          push eax // push 2nd Param
[00000d09] (03) 8b4d08        mov ecx,[ebp+08]
[00000d0c] (01) 51          push ecx // push 1st Param
[00000d0d] (05) e870feffff    call 00000b82 // call H
[00000d12] (03) 83c408        add esp,+08
[00000d15] (02) 85c0          test eax,eax
[00000d17] (02) 7402          jz 00000d1b
[00000d19] (02) ebfe          jmp 00000d19
[00000d1b] (01) 5d          pop ebp
[00000d1c] (01) c3          ret
Size in bytes:(0027) [00000d1c]

```

```

_main()
[00000d22] (01) 55          push ebp
[00000d23] (02) 8bec          mov ebp,esp
[00000d25] (05) 68020d0000    push 00000d02 // push P
[00000d2a] (05) 68020d0000    push 00000d02 // push P
[00000d2f] (05) e8aefffff    call 00000ce2 // call simulate
[00000d34] (03) 83c408        add esp,+08
[00000d37] (01) 50          push eax
[00000d38] (05) 6823030000    push 00000323
[00000d3d] (05) e810f6ffff    call 00000352 // call output
[00000d42] (03) 83c408        add esp,+08
[00000d45] (02) 33c0          xor eax,eax
[00000d47] (01) 5d          pop ebp
[00000d48] (01) c3          ret
Size in bytes:(0039) [00000d48]

```

	machine address	stack address	stack data	machine code	assembly language
...	[0000d22]	[00101851]	[00000000]	55	push ebp
...	[0000d23]	[00101851]	[00000000]	8bec	mov ebp,esp
...	[0000d25]	[0010184d]	[0000d02]	68020d0000	push 0000d02 // push P
...	[0000d2a]	[00101849]	[0000d02]	68020d0000	push 0000d02 // push P
...	[0000d2f]	[00101845]	[0000d34]	e8aeffff	call 0000ce2 // call Simulate
...	[0000ce2]	[00101841]	[00101851]	55	push ebp
...	[0000ce3]	[00101841]	[00101851]	8bec	mov ebp,esp
...	[0000ce5]	[00101841]	[00101851]	8b450c	mov eax,[ebp+0c]
...	[0000ce8]	[0010183d]	[0000d02]	50	push eax
Calling:_P()					
...	[0000ce9]	[00101839]	[0000cec]	ff5508	call dword [ebp+08]
...	[0000d02]	[00101835]	[00101841]	55	push ebp
...	[0000d03]	[00101835]	[00101841]	8bec	mov ebp,esp
...	[0000d05]	[00101835]	[00101841]	8b4508	mov eax,[ebp+08]
...	[0000d08]	[00101831]	[0000d02]	50	push eax
...	[0000d09]	[00101831]	[0000d02]	8b4d08	mov ecx,[ebp+08]
...	[0000d0c]	[0010182d]	[0000d02]	51	push ecx
...	[0000d0d]	[00101829]	[0000d12]	e870feffff	call 0000b82 // call H

Begin Local Halt Decider Simulation at Machine Address:d02

...	[0000d02]	[002118f1]	[002118f5]	55	push ebp
...	[0000d03]	[002118f1]	[002118f5]	8bec	mov ebp,esp
...	[0000d05]	[002118f1]	[002118f5]	8b4508	mov eax,[ebp+08]
...	[0000d08]	[002118ed]	[0000d02]	50	push eax // push P
...	[0000d09]	[002118ed]	[0000d02]	8b4d08	mov ecx,[ebp+08]
...	[0000d0c]	[002118e9]	[0000d02]	51	push ecx // push P
...	[0000d0d]	[002118e5]	[0000d12]	e870feffff	call 0000b82 // call H
...	[0000d02]	[0025c319]	[0025c31d]	55	push ebp
...	[0000d03]	[0025c319]	[0025c31d]	8bec	mov ebp,esp
...	[0000d05]	[0025c319]	[0025c31d]	8b4508	mov eax,[ebp+08]
...	[0000d08]	[0025c315]	[0000d02]	50	push eax // push P
...	[0000d09]	[0025c315]	[0000d02]	8b4d08	mov ecx,[ebp+08]
...	[0000d0c]	[0025c311]	[0000d02]	51	push ecx // push P
...	[0000d0d]	[0025c30d]	[0000d12]	e870feffff	call 0000b82 // call H

Local Halt Decider: Infinite Recursion Detected Simulation Stopped

...	[0000d12]	[00101835]	[00101841]	83c408	add esp,+08
...	[0000d15]	[00101835]	[00101841]	85c0	test eax,eax
...	[0000d17]	[00101835]	[00101841]	7402	jz 0000d1b
...	[0000d1b]	[00101839]	[0000cec]	5d	pop ebp
...	[0000d1c]	[0010183d]	[0000d02]	c3	ret
...	[0000cec]	[00101841]	[00101851]	83c404	add esp,+04
...	[0000cef]	[00101841]	[00101851]	b801000000	mov eax,00000001
...	[0000cf4]	[00101845]	[0000d34]	5d	pop ebp
...	[0000cf5]	[00101849]	[0000d02]	c3	ret
...	[0000d34]	[00101851]	[00000000]	83c408	add esp,+08
...	[0000d37]	[0010184d]	[00000001]	50	push eax
...	[0000d38]	[00101849]	[00000323]	6823030000	push 00000323
---	[0000d3d]	[00101849]	[00000323]	e810f6ffff	call 00000352 // call output

Input\_Halts = 1

...	[0000d42]	[00101851]	[00000000]	83c408	add esp,+08
...	[0000d45]	[00101851]	[00000000]	33c0	xor eax,eax
...	[0000d47]	[00101855]	[00100000]	5d	pop ebp
...	[0000d48]	[00101859]	[000000bc]	c3	ret

Number\_of\_User\_Instructions(48)

Number of Instructions Executed(23742)

Because H only acts as a pure simulator of its input until after its halt status decision has been made it has no behavior that can possibly effect the behavior of its input. Because of this H screens out its own address range in every execution trace that it examines. This is why we never see any instructions of H in any execution trace after an input calls H.

**Halting is provable on the basis that a pure simulation reaches the final state.**  
P reaches its final state.

**Never Halting is provable on the basis that the final state can never be reached.**

(a) We know that the x86 execution trace of the simulation of  $P(P)$  is a pure simulation by comparing it to the source-code of  $P$ . (also see the first paragraph).

(b) We know that whether or not  $H$  aborts its simulation of its input to  $H(P,P)$  that this input would never reach its final state (proved by the x86 execution trace shown above).

Because there are no control flow instructions in the execution trace that would escape the infinite recursion the execution trace proves that a pure simulation of the above input would never reach its final state.

(c) Because input to  $H(P,P)$  would never reach its final state we know that it never halts.

(d) Because it never halts we know that  $H(P,P)$  correctly returns 0 indicating that its input never halts.

**If there is no error in (a)(b)(c)(d) then we know  $H(P,P)=0$  is the correct halt status.**

There may be a very high tendency to reject this latter claim out-of-hand without sufficient review through the human fallibility of bias. If no error exists in steps (a)(b)(c)(d) then we know that  $H(P,P)=0$  is the correct halt status of the input to the input to  $H(P,P)$ .

If  $P(P)$  halts and  $H(P,P)=0$  is the correct halt status of the input to the input to  $H(P,P)$  then we have a paradox rather than a contradiction.

## Simulating partial halt decider H correctly decides that P(P) never halts (V3)

The execution trace of the x86 emulation of P(P) by simulating halt decider H conclusively proves that P cannot possibly ever reach its final state of 0xc3f. This provides complete proof that the input to H never halts thus  $H(P,P)=0$  is correct.

```
// Simplified Linz H (Linz:1990:319)
// Strachey(1965) CPL translated to C
```

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

```
int main()
{
    P((u32)P);
}
```

```
_P()
[0000c25] (01) 55          push ebp
[0000c26] (02) 8bec         mov ebp,esp
[0000c28] (03) 8b4508      mov eax,[ebp+08]
[0000c2b] (01) 50          push eax      // 2nd Param
[0000c2c] (03) 8b4d08      mov ecx,[ebp+08]
[0000c2f] (01) 51          push ecx      // 1st Param
[0000c30] (05) e820fdffff  call 00000955 // call H
[0000c35] (03) 83c408      add esp,+08
[0000c38] (02) 85c0         test eax,eax
[0000c3a] (02) 7402         jz 0000c3e
[0000c3c] (02) ebfe         jmp 0000c3c
[0000c3e] (01) 5d          pop ebp
[0000c3f] (01) c3          ret
Size in bytes:(0027) [0000c3f]
```

```
_main()
[0000c45] (01) 55          push ebp
[0000c46] (02) 8bec         mov ebp,esp
[0000c48] (05) 68250c0000 push 0000c25 // push P
[0000c4d] (05) e8d3ffffff  call 0000c25 // call P(P)
[0000c52] (03) 83c404      add esp,+04
[0000c55] (02) 33c0         xor eax,eax
[0000c57] (01) 5d          pop ebp
[0000c58] (01) c3          ret
Size in bytes:(0020) [0000c58]
```

machine address	stack address	stack data	machine code	assembly language
[0000c45]	[001016d6]	[00000000]	55	push ebp
[0000c46]	[001016d6]	[00000000]	8bec	mov ebp,esp
[0000c48]	[001016d2]	[0000c25]	68250c0000	push 0000c25 // push P
[0000c4d]	[001016ce]	[0000c52]	e8d3ffffff	call 0000c25 // call P <sub>0</sub> (P)
[0000c25]	[001016ca]	[001016d6]	55	push ebp // P <sub>0</sub> begins
[0000c26]	[001016ca]	[001016d6]	8bec	mov ebp,esp
[0000c28]	[001016ca]	[001016d6]	8b4508	mov eax,[ebp+08]
[0000c2b]	[001016c6]	[0000c25]	50	push eax // push P
[0000c2c]	[001016c6]	[0000c25]	8b4d08	mov ecx,[ebp+08]
[0000c2f]	[001016c2]	[0000c25]	51	push ecx // push P
[0000c30]	[001016be]	[0000c35]	e820fdffff	call 00000955 // call H <sub>0</sub> (P <sub>1</sub> ,P <sub>1</sub> )



```

Begin Local Halt Decider Simulation at Machine Address:c25
[0000c25][00211776][0021177a] 55      push ebp      // P1 begins
[0000c26][00211776][0021177a] 8bec     mov ebp,esp
[0000c28][00211776][0021177a] 8b4508   mov eax,[ebp+08]
[0000c2b][00211772][0000c25] 50      push eax     // push P
[0000c2c][00211772][0000c25] 8b4d08   mov ecx,[ebp+08]
[0000c2f][0021176e][0000c25] 51      push ecx     // push P
[0000c30][0021176a][0000c35] e820fdfff call 00000955 // call H1(P2,P2)

[0000c25][0025c19e][0025c1a2] 55      push ebp      // P2 begins
[0000c26][0025c19e][0025c1a2] 8bec     mov ebp,esp
[0000c28][0025c19e][0025c1a2] 8b4508   mov eax,[ebp+08]
[0000c2b][0025c19a][0000c25] 50      push eax     // push P
[0000c2c][0025c19a][0000c25] 8b4d08   mov ecx,[ebp+08]
[0000c2f][0025c196][0000c25] 51      push ecx     // push P
[0000c30][0025c192][0000c35] e820fdfff call 00000955 // call H2(P3,P3)
Local Halt Decider: Infinite Recursion Detected Simulation Stopped

```

In the above computation (zero based addressing)  $H_0$  aborts  $P_1$   
No  $P(P)$  ever stops running unless  $H_0$  aborts its simulation of  $P_1$

The subscripts indicate that a new process context has been created to simulate the virtual machine. It has its own RAM, stack and registers.

```

[0000c35][001016ca][001016d6] 83c408   add esp,+08
[0000c38][001016ca][001016d6] 85c0     test eax,eax
[0000c3a][001016ca][001016d6] 7402     jz 0000c3e
[0000c3e][001016ce][0000c52] 5d      pop ebp
[0000c3f][001016d2][0000c25] c3      ret
[0000c52][001016d6][00000000] 83c404   add esp,+04
[0000c55][001016d6][00000000] 33c0     xor eax,eax
[0000c57][001016da][00100000] 5d      pop ebp
[0000c58][001016de][00000084] c3      ret
Number_of_User_Instructions(34)
Number of Instructions Executed(23729)

```

- (1) H does perform a pure simulation of its input until after it makes its halt status decision.
- (2) It can be verified that this is a pure simulation on the basis that the execution trace does what the x86 source-code of  $P$  specifies.
- (3) Because there are no control flow instructions in the execution trace that can possibly escape the infinite recursion the execution trace proves that a pure simulation of the above input cannot possibly ever reach its final state.
- (4) Therefore  $H$  was correct when it decided that its input never halts.

## Simulating partial halt decider H correctly decides that Infinite\_Loop() never halts

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

int main()
{
    u32 Input_Would_Halt2 = H((u32)Infinite_Loop, (u32)Infinite_Loop);
    Output("Input_Would_Halt2 = ", Input_Would_Halt2);
}
```

```
_Infinite_Loop()
[00000ab0] (01) 55          push ebp
[00000ab1] (02) 8bec        mov ebp,esp
[00000ab3] (02) ebfe        jmp 00000ab3
[00000ab5] (01) 5d          pop ebp
[00000ab6] (01) c3          ret
Size in bytes:(0007) [00000ab6]
```

```
_main()
[00000c00] (01) 55          push ebp
[00000c01] (02) 8bec        mov ebp,esp
[00000c03] (01) 51          push ecx
[00000c04] (05) 68b00a0000 push 00000ab0
[00000c09] (05) 68b00a0000 push 00000ab0
[00000c0e] (05) e84dfdffff call 00000960
[00000c13] (03) 83c408      add esp,+08
[00000c16] (03) 8945fc      mov [ebp-04],eax
[00000c19] (03) 8b45fc      mov eax,[ebp-04]
[00000c1c] (01) 50          push eax
[00000c1d] (05) 684b030000 push 0000034b
[00000c22] (05) e859f7ffff call 00000380
[00000c27] (03) 83c408      add esp,+08
[00000c2a] (02) 33c0      xor eax,eax
[00000c2c] (02) 8be5      mov esp,ebp
[00000c2e] (01) 5d          pop ebp
[00000c2f] (01) c3          ret
Size in bytes:(0048) [00000c2f]
```

## Execution Trace of H(Infinite\_Loop, Infinite\_Loop)

machine address	stack address	stack data	machine code	assembly language
[00000c00]	[00101693]	[00000000]	55	push ebp
[00000c01]	[00101693]	[00000000]	8bec	mov ebp,esp
[00000c03]	[0010168f]	[00000000]	51	push ecx
[00000c04]	[0010168b]	[00000ab0]	68b00a0000	push 00000ab0
[00000c09]	[00101687]	[00000ab0]	68b00a0000	push 00000ab0
[00000c0e]	[00101683]	[00000c13]	e84dfdffff	call 00000960

Begin Local Halt Decider Simulation at Machine Address:ab0

[00000ab0]	[00211733]	[00211737]	55	push ebp
[00000ab1]	[00211733]	[00211737]	8bec	mov ebp,esp
[00000ab3]	[00211733]	[00211737]	ebfe	jmp 00000ab3
[00000ab3]	[00211733]	[00211737]	ebfe	jmp 00000ab3

Local Halt Decider: Infinite Loop Detected Simulation Stopped

```

[0000c13][0010168f][00000000] 83c408    add esp,+08
[0000c16][0010168f][00000000] 8945fc    mov [ebp-04],eax
[0000c19][0010168f][00000000] 8b45fc    mov eax,[ebp-04]
[0000c1c][0010168b][00000000] 50        push eax
[0000c1d][00101687][0000034b] 684b030000 push 0000034b
[0000c22][00101687][0000034b] e859f7ffff call 00000380
Input_would_Halt2 = 0
[0000c27][0010168f][00000000] 83c408    add esp,+08
[0000c2a][0010168f][00000000] 33c0      xor eax,eax
[0000c2c][00101693][00000000] 8be5      mov esp,ebp
[0000c2e][00101697][00100000] 5d        pop ebp
[0000c2f][0010169b][00000050] c3        ret
Number_of_User_Instructions(21)
Number of Instructions Executed(640)

```

## Simulating partial halt decider H decides that Infinite\_Recursion() never halts

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

int main()
{
    u32 Input_Halts = H((u32)Infinite_Recursion, 3);
    Output("Input_Halts = ", Input_Halts);
}
```

```
_Infinite_Recursion()
[0000ac6] (01) 55          push ebp
[0000ac7] (02) 8bec         mov ebp,esp
[0000ac9] (03) 8b4508      mov eax,[ebp+08]
[0000acc] (01) 50          push eax
[0000acd] (05) e8f4ffffff  call 0000ac6
[0000ad2] (03) 83c404      add esp,+04
[0000ad5] (01) 5d          pop ebp
[0000ad6] (01) c3          ret
Size in bytes:(0017) [0000ad6]
```

```
_main()
[0000c46] (01) 55          push ebp
[0000c47] (02) 8bec         mov ebp,esp
[0000c49] (01) 51          push ecx
[0000c4a] (02) 6a03         push +03
[0000c4c] (05) 68c60a0000  push 0000ac6
[0000c51] (05) e810fdffff  call 0000966
[0000c56] (03) 83c408      add esp,+08
[0000c59] (03) 8945fc      mov [ebp-04],eax
[0000c5c] (03) 8b45fc      mov eax,[ebp-04]
[0000c5f] (01) 50          push eax
[0000c60] (05) 6857030000  push 0000357
[0000c65] (05) e81cf7ffff  call 0000386
[0000c6a] (03) 83c408      add esp,+08
[0000c6d] (02) 33c0         xor eax,eax
[0000c6f] (02) 8be5         mov esp,ebp
[0000c71] (01) 5d          pop ebp
[0000c72] (01) c3          ret
Size in bytes:(0045) [0000c72]
```

## Execution Trace of H(Infinite\_Recursion, 3)

machine address	stack address	stack data	machine code	assembly language
[0000c46]	[001016fa]	[00000000]	55	push ebp
[0000c47]	[001016fa]	[00000000]	8bec	mov ebp,esp
[0000c49]	[001016f6]	[00000000]	51	push ecx
[0000c4a]	[001016f2]	[00000003]	6a03	push +03
[0000c4c]	[001016ee]	[00000ac6]	68c60a0000	push 0000ac6
[0000c51]	[001016ea]	[00000c56]	e810fdffff	call 0000966

```

Begin Local Halt Decider Simulation at Machine Address:ac6
[0000ac6][0021179a][0021179e] 55      push ebp
[0000ac7][0021179a][0021179e] 8bec   mov ebp,esp
[0000ac9][0021179a][0021179e] 8b4508 mov eax,[ebp+08]
[0000acc][00211796][00000003] 50     push eax
[0000acd][00211792][00000ad2] e8f4ffff call 0000ac6
[0000ac6][0021178e][0021179a] 55     push ebp
[0000ac7][0021178e][0021179a] 8bec   mov ebp,esp
[0000ac9][0021178e][0021179a] 8b4508 mov eax,[ebp+08]
[0000acc][0021178a][00000003] 50     push eax
[0000acd][00211786][00000ad2] e8f4ffff call 0000ac6
Local Halt Decider: Infinite Recursion Detected Simulation Stopped

```

\_Infinite\_Recursion() calls itself recursively with the same input. It has no escape from this infinite recursion. H recognizes this infinite behavior pattern, aborts its simulation of \_Infinite\_Recursion() and reports that this input never halts.

```

[0000c56][001016f6][00000000] 83c408 add esp,+08
[0000c59][001016f6][00000000] 8945fc mov [ebp-04],eax
[0000c5c][001016f6][00000000] 8b45fc mov eax,[ebp-04]
[0000c5f][001016f2][00000000] 50     push eax
[0000c60][001016ee][00000357] 6857030000 push 00000357
[0000c65][001016ee][00000357] e81cf7ffff call 00000386
Input_Halts = 0
[0000c6a][001016f6][00000000] 83c408 add esp,+08
[0000c6d][001016f6][00000000] 33c0   xor eax,eax
[0000c6f][001016fa][00000000] 8be5   mov esp,ebp
[0000c71][001016fe][00100000] 5d     pop ebp
[0000c72][00101702][00000068] c3     ret
Number_of_User_Instructions(27)
Number of Instructions Executed(1240)

```

#### Infinite recursion detection criteria:

If the execution trace of function X() called by function Y() shows:

- (1) Function X() is called twice in sequence from the same machine address of Y().
- (2) With the same parameters to X().
- (3) With no conditional branch or indexed jump instructions in Y().
- (4) With no function call returns from X().

then the function call from Y() to X() is infinitely recursive.

## Simulating partial halt decider H decides that Factorial(3) halts

```
int Factorial(int n)
{
    output("Factorial(n)",n);
    if (n > 1)
        return n * Factorial(n - 1);
    else
        return 1;
}

int main()
{
    output("Input_Halts = ", H(Factorial, 3));
}
```

```
_Factorial()
[0000de2] (01) 55          push ebp
[0000de3] (02) 8bec         mov ebp,esp
[0000de5] (03) 8b4508      mov eax,[ebp+08]
[0000de8] (01) 50          push eax
[0000de9] (05) 6813030000 push 00000313
[0000dee] (05) e85ff5ffff  call 00000352
[0000df3] (03) 83c408      add esp,+08
[0000df6] (04) 837d0801    cmp dword [ebp+08],+01
[0000dfa] (02) 7e17        jng 0000e13
[0000dfc] (03) 8b4d08      mov ecx,[ebp+08]
[0000dff] (03) 83e901      sub ecx,+01
[0000e02] (01) 51          push ecx
[0000e03] (05) e8daffffff  call 0000de2
[0000e08] (03) 83c404      add esp,+04
[0000e0b] (04) 0faf4508    imul eax,[ebp+08]
[0000e0f] (02) eb07        jmp 0000e18
[0000e11] (02) eb05        jmp 0000e18
[0000e13] (05) b801000000 mov eax,00000001
[0000e18] (01) 5d          pop ebp
[0000e19] (01) c3          ret
Size in bytes:(0056) [0000e19]
```

```
_main()
[0000ea2] (01) 55          push ebp
[0000ea3] (02) 8bec         mov ebp,esp
[0000ea5] (02) 6a03        push +03
[0000ea7] (05) 68e20d0000 push 0000de2
[0000eac] (05) e821feffff  call 0000cd2
[0000eb1] (03) 83c408      add esp,+08
[0000eb4] (01) 50          push eax
[0000eb5] (05) 6823030000 push 0000323
[0000eba] (05) e893f4ffff  call 0000352
[0000ebf] (03) 83c408      add esp,+08
[0000ec2] (02) 33c0        xor eax,eax
[0000ec4] (01) 5d          pop ebp
[0000ec5] (01) c3          ret
Size in bytes:(0036) [0000ec5]
```

	machine address	stack address	stack data	machine code	assembly language
...	[0000ea2]	[00101ae7]	[00000000]	55	push ebp
...	[0000ea3]	[00101ae7]	[00000000]	8bec	mov ebp,esp
...	[0000ea5]	[00101ae3]	[00000003]	6a03	push +03
...	[0000ea7]	[00101adf]	[00000de2]	68e20d0000	push 00000de2
...	[0000eac]	[00101adb]	[00000eb1]	e821feffff	call 0000cd2
Begin Local Halt Decider Simulation at Machine Address:de2					
...	[0000de2]	[00211b87]	[00211b8b]	55	push ebp
...	[0000de3]	[00211b87]	[00211b8b]	8bec	mov ebp,esp
...	[0000de5]	[00211b87]	[00211b8b]	8b4508	mov eax,[ebp+08]
...	[0000de8]	[00211b83]	[00000003]	50	push eax
...	[0000de9]	[00211b7f]	[00000313]	6813030000	push 00000313
---	[0000dee]	[00211b7f]	[00000313]	e85ff5ffff	call 00000352
Factorial(n)3					
...	[0000df3]	[00211b87]	[00211b8b]	83c408	add esp,+08
...	[0000df6]	[00211b87]	[00211b8b]	837d0801	cmp dword [ebp+08],+01
...	[0000dfa]	[00211b87]	[00211b8b]	7e17	jng 0000e13
...	[0000dfc]	[00211b87]	[00211b8b]	8b4d08	mov ecx,[ebp+08]
...	[0000dff]	[00211b87]	[00211b8b]	83e901	sub ecx,+01
...	[0000e02]	[00211b83]	[00000002]	51	push ecx
...	[0000e03]	[00211b7f]	[00000e08]	e8daffffff	call 0000de2
...	[0000de2]	[00211b7b]	[00211b87]	55	push ebp
...	[0000de3]	[00211b7b]	[00211b87]	8bec	mov ebp,esp
...	[0000de5]	[00211b7b]	[00211b87]	8b4508	mov eax,[ebp+08]
...	[0000de8]	[00211b77]	[00000002]	50	push eax
...	[0000de9]	[00211b73]	[00000313]	6813030000	push 00000313
---	[0000dee]	[00211b73]	[00000313]	e85ff5ffff	call 00000352
Factorial(n)2					
...	[0000df3]	[00211b7b]	[00211b87]	83c408	add esp,+08
...	[0000df6]	[00211b7b]	[00211b87]	837d0801	cmp dword [ebp+08],+01
...	[0000dfa]	[00211b7b]	[00211b87]	7e17	jng 0000e13
...	[0000dfc]	[00211b7b]	[00211b87]	8b4d08	mov ecx,[ebp+08]
...	[0000dff]	[00211b7b]	[00211b87]	83e901	sub ecx,+01
...	[0000e02]	[00211b77]	[00000001]	51	push ecx
...	[0000e03]	[00211b73]	[00000e08]	e8daffffff	call 0000de2
...	[0000de2]	[00211b6f]	[00211b7b]	55	push ebp
...	[0000de3]	[00211b6f]	[00211b7b]	8bec	mov ebp,esp
...	[0000de5]	[00211b6f]	[00211b7b]	8b4508	mov eax,[ebp+08]
...	[0000de8]	[00211b6b]	[00000001]	50	push eax
...	[0000de9]	[00211b67]	[00000313]	6813030000	push 00000313
---	[0000dee]	[00211b67]	[00000313]	e85ff5ffff	call 00000352
Factorial(n)1					
...	[0000df3]	[00211b6f]	[00211b7b]	83c408	add esp,+08
...	[0000df6]	[00211b6f]	[00211b7b]	837d0801	cmp dword [ebp+08],+01
...	[0000dfa]	[00211b6f]	[00211b7b]	7e17	jng 0000e13
...	[0000e13]	[00211b6f]	[00211b7b]	b801000000	mov eax,00000001
...	[0000e18]	[00211b73]	[00000e08]	5d	pop ebp
...	[0000e19]	[00211b77]	[00000001]	c3	ret
...	[0000eb1]	[00101ae7]	[00000000]	83c408	add esp,+08
...	[0000eb4]	[00101ae3]	[00000001]	50	push eax
...	[0000eb5]	[00101adf]	[00000323]	6823030000	push 00000323
---	[0000eba]	[00101adf]	[00000323]	e893f4ffff	call 00000352
Input Halts = 1					
...	[0000ebf]	[00101ae7]	[00000000]	83c408	add esp,+08
...	[0000ec2]	[00101ae7]	[00000000]	33c0	xor eax,eax
...	[0000ec4]	[00101aeb]	[00100000]	5d	pop ebp
...	[0000ec5]	[00101aef]	[000000c8]	c3	ret
Number_of_User_Instructions(51)					
Number of Instructions Executed(3714)					

# Strachey's Impossible Program

To the Editor,  
The Computer Journal.

## An impossible program

Sir,

A well-known piece of folk-lore among programmers holds that it is impossible to write a program which can examine any other program and tell, in every case, if it will terminate or get into a closed loop when it is run. I have never actually seen a proof of this in print, and though Alan Turing once gave me a verbal proof (in a railway carriage on the way to a Conference at the NPL in 1953), I unfortunately and promptly forgot the details. This left me with an uneasy feeling that the proof must be long or complicated, but in fact it is so short and simple that it may be of interest to casual readers. The version below uses CPL, but not in any essential way.

Suppose  $T[R]$  is a Boolean function taking a routine (or program)  $R$  with no formal or free variables as its argument and that for all  $R$ ,  $T[R]$  — True if  $R$  terminates if run and that  $T[R] = \text{False}$  if  $R$  does not terminate. Consider the routine  $P$  defined as follows

```
rec routine P
  §L:if T[P] go to L
  Return §
```

If  $T[P] = \text{True}$  the routine  $P$  will loop, and it will only terminate if  $T[P] = \text{False}$ . In each case  $T[P]$  has exactly the wrong value, and this contradiction shows that the function  $T$  cannot exist.

Yours faithfully,  
C. STRACHEY.

Churchill College,  
Cambridge.

**Strachey, C 1965.** An impossible program The Computer Journal, Volume 7, Issue 4, January 1965, Page 313, <https://doi.org/10.1093/comjnl/7.4.313>



# Peter Linz $\hat{H}$ applied to the Turing machine description of itself: $\langle \hat{H} \rangle$

The following simplifies the syntax for the definition of the Linz Turing machine  $\hat{H}$ , it is now a single machine with a single start state. The halt decider is embedded at state  $\hat{H}.qx$ .

$\hat{H}.q_0 wM \vdash^* \hat{H}.qx wM wM \vdash^* \hat{H}.qy \infty$   
 if M applied to wM halts, and

$\hat{H}.q_0 wM \vdash^* \hat{H}.qx wM wM \vdash^* \hat{H}.qn$   
 if M applied to wM does not halt // M refers to the TM of the first wM parameter to  $\hat{H}.qx$

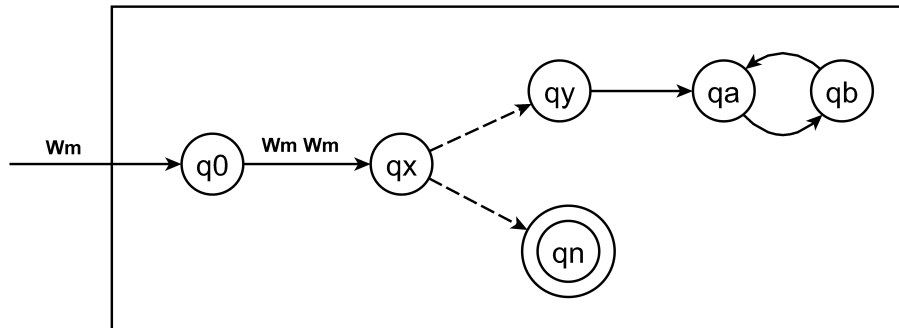


Figure 12.3 Turing Machine  $\hat{H}$

To provide a sketch of the idea of how a simulating halt decider would analyze the Peter Linz  $\hat{H}$  applied to its own Turing machine description we start by examining the behavior of an ordinary UTM.

When we hypothesize that the halt decider embedded in  $\hat{H}$  is simply a UTM then it seems that when the Peter Linz  $\hat{H}$  is applied to its own Turing machine description  $\langle \hat{H} \rangle$  this specifies a computation that never halts.

$\hat{H}_0.q_0$  copies its input  $\langle \hat{H}_1 \rangle$  to  $\langle \hat{H}_2 \rangle$  then  $\hat{H}_0.qx$  simulates  $\hat{H}_1$  with the  $\langle \hat{H}_2 \rangle$  copy then  $\hat{H}_1.q_0$  copies its input  $\langle \hat{H}_2 \rangle$  to  $\langle \hat{H}_3 \rangle$  then  $\hat{H}_1.qx$  simulates  $\hat{H}_2$  with the  $\langle \hat{H}_3 \rangle$  copy then  $\hat{H}_2.q_0$  copies its input  $\langle \hat{H}_3 \rangle$  to  $\langle \hat{H}_4 \rangle$  then  $\hat{H}_2.qx$  simulates  $\hat{H}_3$  with the  $\langle \hat{H}_4 \rangle$  copy then ...

This is expressed in figure 12.4 as a cycle from qx to q0 to qx.

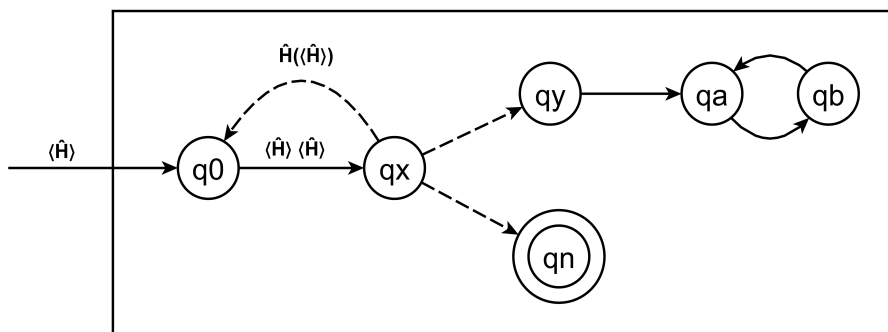


Figure 12.4 Turing Machine  $\hat{H}$  applied to  $\langle \hat{H} \rangle$  input

Next we examine the behavior of  $\hat{H}$  applied to its own Turing machine description:  $\langle \hat{H} \rangle$  when the halt decider at  $\hat{H}.qx$  bases its halt status decision on simulating its input.

Turing machine  $\hat{H}$  is applied to its input  $\langle \hat{H} \rangle$ . It copies this input such that this input and the copy of this input become the first and second parameters to the simulating halt decider at  $\hat{H}.qx$ . When  $\hat{H}.qx \langle \hat{H} \rangle \langle \hat{H} \rangle$  decides that the simulation of its first parameter on the input of its second parameter never halt it correctly transitions to its own final state of  $\hat{H}.qn$ .

Within the hypothesis that the internal halt decider embedded within  $\hat{H}$  simulates its input when  $\hat{H}$  is applied to its own Turing machine description  $\langle \hat{H} \rangle$  then we can see that this derives infinitely nested simulation that must be aborted.  $\hat{H}$  applied to  $\langle \hat{H} \rangle$  specifies an infinite cycle from  $\hat{H}.qx$  to  $\hat{H}.q0$  all the time that  $\hat{H}.qx$  remains a pure simulator of its input.

The fact that  $\hat{H}$  applied to  $\langle \hat{H} \rangle$  transitions to its final state of  $\hat{H}.qn$  and halts does not nullify the fact that  $\hat{H}.qx \langle \hat{H} \rangle \langle \hat{H} \rangle$  does correctly decide that its input never halts.

### **Self-Evident-Truth (premise[1])**

When the pure simulation of a machine on its input never halts we know that the execution of this machine on its input never halts. **Derived from  $UTM(P,I) \equiv P(I)$ .**

### **Self-Evident-Truth (premise[2])**

The  $\langle \hat{H} \rangle \langle \hat{H} \rangle$  input to the embedded simulating halt decider at  $\hat{H}.qx$  cannot possibly reach its final state and halt whether or not the simulation is ever aborted.

### **∴ Sound Deductive Conclusion**

The embedded simulating halt decider at  $\hat{H}.qx$  correctly decides its input:  $\langle \hat{H} \rangle \langle \hat{H} \rangle$  is a computation that never halts.

$\hat{H}.q0 \langle \hat{H} \rangle$  specifies an infinite chain of invocations that is terminated at its third invocation. The first invocation of  $\hat{H}.qx \langle \hat{H} \rangle, \langle \hat{H} \rangle$  is the first element of an infinite chain of invocations.

That the first element of this infinite chain terminates after its third element has been terminated does not entail that this first element is an actual terminating computation. For the first element to be an actual terminating computation it must terminate without any of the elements of the infinite chain of invocations being terminated.

Copyright 2016-2021 PL Olcott

**Strachey, C 1965.** An impossible program The Computer Journal, Volume 7, Issue 4, January 1965, Page 313, <https://doi.org/10.1093/comjnl/7.4.313>

**Linz, Peter 1990.** An Introduction to Formal Languages and Automata. Lexington/Toronto: D. C. Heath and Company. (318-320)

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

**Theorem 12.1**

There does not exist any Turing machine  $H$  that behaves as required by Definition 12.1. The halting problem is therefore undecidable.

**Proof:** We assume the contrary, namely that there exists an algorithm, and consequently some Turing machine  $H$ , that solves the halting problem. The input to  $H$  will be the description (encoded in some form) of  $M$ , say  $w_M$ , as well as the input  $w$ . The requirement is then that, given any  $(w_M, w)$ , the Turing machine  $H$  will halt with either a yes or no answer. We achieve this by asking that  $H$  halt in one of two corresponding final states, say,  $q_y$  or  $q_n$ . The situation can be visualized by a block diagram like Figure 12.1. The intent of this diagram is to indicate that, if  $M$  is started in state  $q_0$  with input  $(w_M, w)$ , it will eventually halt in state  $q_y$  or  $q_n$ . As required by Definition 12.1, we want  $H$  to operate according to the following rules:

$$q_0 w_M w \vdash^* H x_1 q_y x_2,$$

if  $M$  applied to  $w$  halts, and

$$q_0 w_M w \vdash^* H y_1 q_n y_2,$$

if  $M$  applied to  $w$  does not halt.

**Figure 12.1**

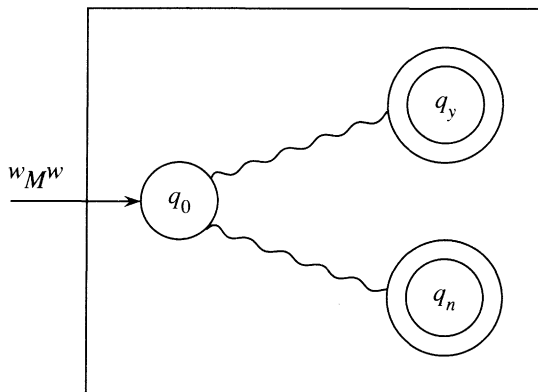
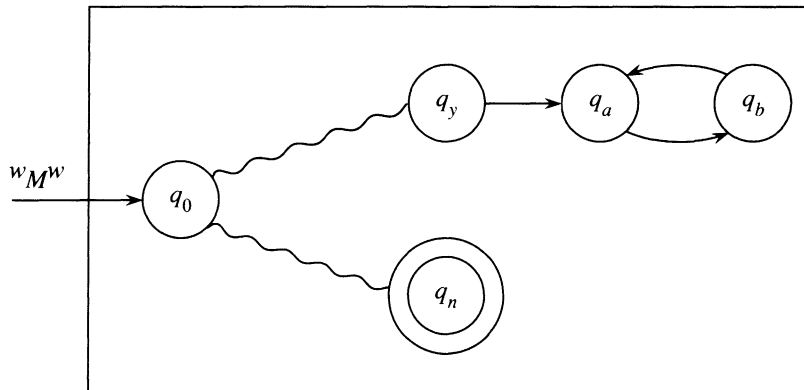


Figure 12.2



Next, we modify  $H$  to produce a Turing machine  $H'$  with the structure shown in Figure 12.2. With the added states in Figure 12.2 we want to convey that the transitions between state  $q_y$  and the new states  $q_a$  and  $q_b$  are to be made, regardless of the tape symbol, in such a way that the tape remains unchanged. The way this is done is straightforward. Comparing  $H$  and  $H'$  we see that, in situations where  $H$  reaches  $q_y$  and halts, the modified machine  $H'$  will enter an infinite loop. Formally, the action of  $H'$  is described by

$$q_0 w_M w \vdash_{H'}^* \infty,$$

if  $M$  applied to  $w$  halts, and

$$q_0 w_M w \vdash_{H'}^* y_1 q_n y_2,$$

if  $M$  applied to  $w$  does not halt.

From  $H'$  we construct another Turing machine  $\hat{H}$ . This new machine takes as input  $w_M$ , copies it, and then behaves exactly like  $H'$ . Then the action of  $\hat{H}$  is such that

$$q_0 w_M \vdash_{\hat{H}}^* q_0 w_M w_M \vdash_{\hat{H}}^* \infty,$$

if  $M$  applied to  $w_M$  halts, and

$$q_0 w_M \vdash_{\hat{H}}^* q_0 w_M w_M \vdash_{\hat{H}}^* y_1 q_n y_2,$$

if  $M$  applied to  $w_M$  does not halt.

Now  $\hat{H}$  is a Turing machine, so that it will have some description in  $\Sigma^*$ , say  $\hat{w}$ . This string, in addition to being the description of  $\hat{H}$  can also be used as input string. We can therefore legitimately ask what would happen if  $\hat{H}$  is applied to  $\hat{w}$ . From the above, identifying  $M$  with  $\hat{H}$ , we get

$$q_0\hat{w} \vdash^* \hat{H}\infty,$$

if  $\hat{H}$  applied to  $\hat{w}$  halts, and

$$q_0\hat{w} \vdash^* \hat{H}y_1q_ny_2,$$

if  $\hat{H}$  applied to  $\hat{w}$  does not halt. This is clearly nonsense. The contradiction tells us that our assumption of the existence of  $H$ , and hence the assumption of the decidability of the halting problem, must be false. ■