Halting problem undecidability and infinitely nested simulation

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 to the confounding input.

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.

```
procedure compute_g(i):
  if f(i, i) == 0 then
    return 0
  else
    loop forever     // (Wikipedia:Halting Problem)
```

The x86utm operating system was created so that the halting problem could be examined concretely in the high level language of C. UTM tape elements are 32-bit unsigned integers. 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 does not halt.

A simulating halt decider must abort the simulation of every input that never halts.
For H to recognize the infinitely repeating pattern of P it only needs to see that same thing that humans see when they examine the x86 execution trace of the simulation of P.

```
// Simplified Linz A (Linz:1990:319)
void P(u32 x)
{
  u32 Input_Halts = H(x, x);
  if (Input_Halts)
    HERE: goto HERE;
}


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

(a) We can know that the simulation of the input to H(P,P) never halts without being aborted with 100% perfect certainty on the basis of its x86 execution trace. (shown below).

(b) From (a) we can know with 100% perfect certainty that simulating halt decider H must abort its simulation of its input (P,P).

(c) From (a) and (b) we can know with 100% perfect certainty that simulating halt decider H correctly reports that its input: (P,P) never halts.

Each of the above steps can be verified as completely true entirely on the basis of the meaning of its words.

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]

================================
...[00000c00][00101693][00000000](01)  55              push ebp
...[00000c01][00101693][00000000](02)  8bec            mov ebp,esp
...[00000c03][0010168f][00000000](01)  51              push ecx
...[00000c04][0010168b][00000ab0](05)  68b00a0000      push 00000ab0
...[00000c09][00101687][00000ab0](05)  68b00a0000      push 00000ab0
...[00000c0e][00101683][00000c13](05)  e84dfdffff      call 00000960
Begin Local Halt Decider Simulation at Machine Address:ab0
...[00000ab0][00211733][00211737](01)  55              push ebp
...[00000ab1][00211733][00211737](02)  8bec            mov ebp,esp
...[00000ab3][00211733][00211737](02)  ebfe            jmp 00000ab3
...[00000ab3][00211733][00211737](02)  ebfe            jmp 00000ab3
Local Halt Decider: Infinite Loop Detected Simulation Stopped
...[00000c13][0010168f][00000000](03)  83c408          add esp,+08
...[00000c16][0010168f][00000000](03)  8945fc          mov [ebp-04],eax
...[00000c19][0010168f][00000000](03)  8b45fc          mov eax,[ebp-04]
...[00000c1c][0010168b][00000000](01)  50              push eax
...[00000c1d][00101687][0000034b](05)  684b030000      push 0000034b
---[00000c22][00101687][0000034b](05)  e859f7ffff      call 00000380
Input_Would_Halt2 = 0
...[00000c27][0010168f][00000000](03)  83c408          add esp,+08
...[00000c2a][0010168f][00000000](02)  33c0            xor eax,eax
...[00000c2c][00101693][00000000](02)  8be5            mov esp,ebp
...[00000c2e][00101697][00100000](01)  5d              pop ebp
...[00000c2f][0010169b][00000050](01)  c3              ret
Number_of_User_Instructions(21)
Number of Instructions Executed(640)
```

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

```
// Simplified Linz A (Linz:1990:319)
void P(u32 x)
{
  u32 Input_Halts = H(x, x);
  if (Input_Halts)
    HERE: goto HERE;
}

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

_P()
[00000b1a](01)  55              push ebp
[00000b1b](02)  8bec            mov ebp,esp
[00000b1d](01)  51              push ecx
[00000b1e](03)  8b4508          mov eax,[ebp+08]
[00000b21](01)  50              push eax
[00000b22](03)  8b4d08          mov ecx,[ebp+08]
[00000b25](01)  51              push ecx
[00000b26](05)  e81ffeffff      call 0000094a
[00000b2b](03)  83c408          add esp,+08
[00000b2e](03)  8945fc          mov [ebp-04],eax
[00000b31](04)  837dfc00        cmp dword [ebp-04],+00
[00000b35](02)  7402            jz 00000b39
[00000b37](02)  ebfe            jmp 00000b37
[00000b39](02)  8be5            mov esp,ebp
[00000b3b](01)  5d              pop ebp
[00000b3c](01)  c3              ret
Size in bytes:(0035) [00000b3c]

_main()
[00000bda](01)  55              push ebp
[00000bdb](02)  8bec            mov ebp,esp
[00000bdd](01)  51              push ecx
[00000bde](05)  681a0b0000      push 00000b1a
[00000be3](05)  681a0b0000      push 00000b1a
[00000be8](05)  e85dfdffff      call 0000094a
[00000bed](03)  83c408          add esp,+08
[00000bf0](03)  8945fc          mov [ebp-04],eax
[00000bf3](03)  8b45fc          mov eax,[ebp-04]
[00000bf6](01)  50              push eax
[00000bf7](05)  683b030000      push 0000033b
[00000bfc](05)  e869f7ffff      call 0000036a
[00000c01](03)  83c408          add esp,+08
[00000c04](02)  33c0            xor eax,eax
[00000c06](02)  8be5            mov esp,ebp
[00000c08](01)  5d              pop ebp
[00000c09](01)  c3              ret
Size in bytes:(0048) [00000c09]
```

Columns
(1) Machine address of instruction
(2) Machine address of top of stack
(3) Value of top of stack after instruction executed
(4) Machine language bytes
(5) Assembly language text

```
================================
...[00000bda][00101647][00000000](01)  55              push ebp
...[00000bdb][00101647][00000000](02)  8bec            mov ebp,esp
...[00000bdd][00101643][00000000](01)  51              push ecx
...[00000bde][0010163f][00000b1a](05)  681a0b0000      push 00000b1a
...[00000be3][0010163b][00000b1a](05)  681a0b0000      push 00000b1a
...[00000be8][00101637][00000bed](05)  e85dfdffff      call 0000094a
Begin Local Halt Decider Simulation at Machine Address:b1a
...[00000b1a][002116e7][002116eb](01)  55              push ebp
...[00000b1b][002116e7][002116eb](02)  8bec            mov ebp,esp
...[00000b1d][002116e3][002016b7](01)  51              push ecx
...[00000b1e][002116e3][002016b7](03)  8b4508          mov eax,[ebp+08]
...[00000b21][002116df][00000b1a](01)  50              push eax
...[00000b22][002116df][00000b1a](03)  8b4d08          mov ecx,[ebp+08]
...[00000b25][002116db][00000b1a](01)  51              push ecx
...[00000b26][002116d7][00000b2b](05)  e81ffeffff      call 0000094a
...[00000b1a][0025c10f][0025c113](01)  55              push ebp
...[00000b1b][0025c10f][0025c113](02)  8bec            mov ebp,esp
...[00000b1d][0025c10b][0024c0df](01)  51              push ecx
...[00000b1e][0025c10b][0024c0df](03)  8b4508          mov eax,[ebp+08]
...[00000b21][0025c107][00000b1a](01)  50              push eax
...[00000b22][0025c107][00000b1a](03)  8b4d08          mov ecx,[ebp+08]
...[00000b25][0025c103][00000b1a](01)  51              push ecx
...[00000b26][0025c0ff][00000b2b](05)  e81ffeffff      call 0000094a
Local Halt Decider: Infinite Recursion Detected Simulation Stopped
```

In the above 16 instructions of the simulation of P(P) we can see that the first 8 instructions of P are repeated. The end of this sequence of 8 instructions is a call 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 8 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) would never halt on its input.

```
...[00000bed][00101643][00000000](03)  83c408          add esp,+08
...[00000bf0][00101643][00000000](03)  8945fc          mov [ebp-04],eax
...[00000bf3][00101643][00000000](03)  8b45fc          mov eax,[ebp-04]
...[00000bf6][0010163f][00000000](01)  50              push eax
...[00000bf7][0010163b][0000033b](05)  683b030000      push 0000033b
---[00000bfc][0010163b][0000033b](05)  e869f7ffff      call 0000036a
Input_Halts = 0
...[00000c01][00101643][00000000](03)  83c408          add esp,+08
...[00000c04][00101643][00000000](02)  33c0            xor eax,eax
...[00000c06][00101647][00000000](02)  8be5            mov esp,ebp
...[00000c08][0010164b][00100000](01)  5d              pop ebp
...[00000c09][0010164f][00000080](01)  c3              ret
Number_of_User_Instructions(33)
Number of Instructions Executed(26452)
```

Peter Linz Ĥ applied to the Turing machine description of itself: ŵ

When we assume that the halt decider embedded in Ĥ is simply a UTM does this define a computation that never halts when Ĥ is applied to its own Turing machine description?

The following simplifies the syntax for the definition of the Linz Turing machine Ĥ, it is now a single machine with a single start state. The halt decider is embedded at state Ĥ.qx.

Ĥ.q0 wM ⊢* Ĥ.qx wM wM ⊢* Ĥ.qy ∞
if M applied to wM halts, and

Ĥ.q0 wM ⊢* Ĥ.qx wM wM ⊢* Ĥ.qn
if M applied to wM does not halt

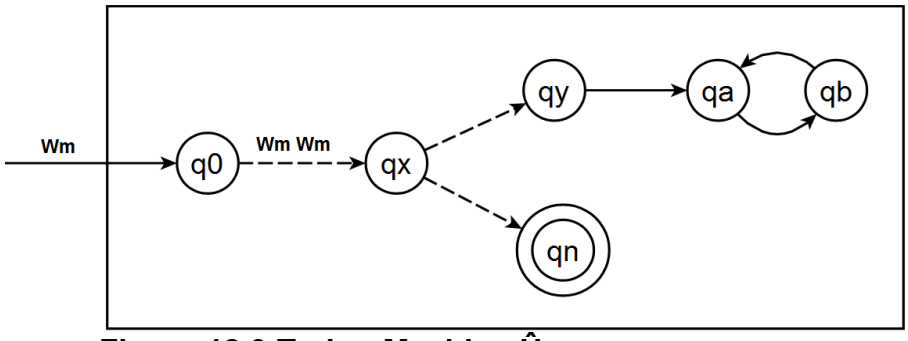


Figure 12.3 Turing Machine Ĥ

Ĥ.q0 copies its input then Ĥ.qx simulates this input with the copy then
Ĥ.q0 copies its input then Ĥ.qx simulates this input with the copy then
Ĥ.q0 copies its input then Ĥ.qx simulates this input with the copy then...
This is expressed in figure 12.4 as a cycle from qx to q0 to qx.

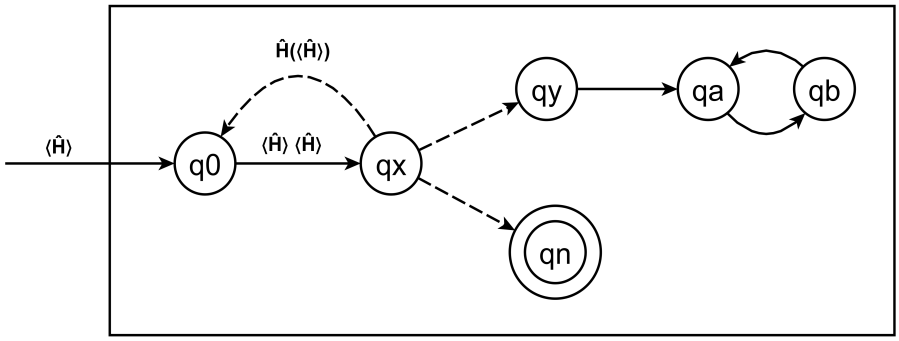


Figure 12.4 Turing Machine Ĥ applied to ⟨Ĥ⟩ input

Within the hypothesis that the internal halt decider embedded within Ĥ simulates its input Ĥ applied to its own Turing machine description ŵ seems to derive infinitely nested simulation, unless this simulation is aborted.

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

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:

$$q_0 w_M w \overset{*}{\vdash}_H x_1 q_y x_2,$$

if M applied to w halts, and

$$q_0 w_M w \overset{*}{\vdash}_H y_1 q_n y_2,$$

if M applied to w does not halt.

Figure 12.1

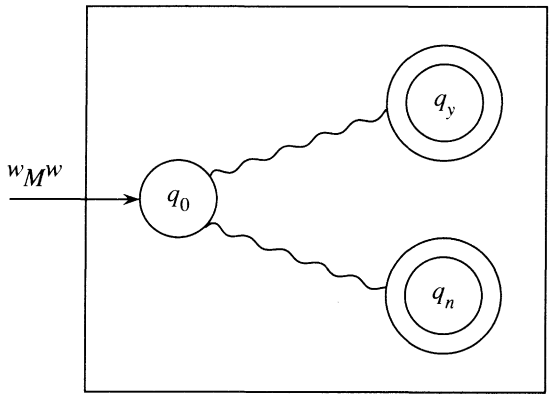
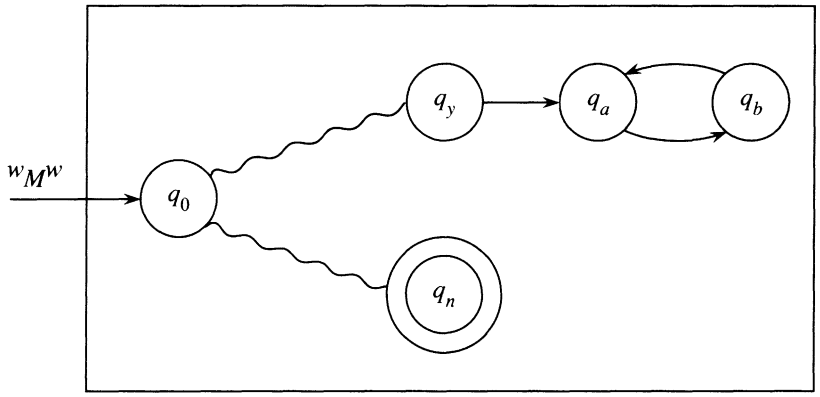


Figure 12.2



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 Σ^*, 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

$$q_0\hat{w} \vdash^*_{\hat{H}} y_1 q_n y_2,$$

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. ■