

Halting problem undecidability and infinitely nested simulation

When halting is defined as any computation that halts without ever having its simulation aborted then it can be understood that partial halt decider H correctly decides that its input does not halt on the simplified version of the Linz \hat{H} .

When this simplified concrete example is fully understood then the exact same reasoning is applied to the actual Linz \hat{H} correctly deciding that it would never halt when applied to its own Turing machine description.

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 examines the behavior of the x86 emulation of its input. 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.

```
int Simulate(u32 P, u32 I)
{
    ((void (*)(u32))P)(I);
    return 1;
}

// Simplified Linz  $\hat{H}$  (Linz:1990:319)
void H_Hat(u32 P)
{
    u32 Input_Halts = H(P, P); // Linz H as a partial halt decider
    if (Input_Halts)
        HERE: goto HERE;
}

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

A simulating halt decider H is a Universal Turing Machine (UTM) that has been adapted to decide whether or not its input halts. H simulates the execution of its inputs exactly as if it was simply a UTM. After H simulates each instruction of its input it examines the full execution trace of this input. When an execution trace matches an infinite execution behavior pattern H aborts the simulation of this input and reports that this input does not halt.

A simulating halt decider that never stops simulating its input is simply a simulator on this input. If $H()$ never stopped simulating $H_Hat()$ then the halting behavior of $H_Hat()$ would be the same as if $H_Hat()$ was called by $Simulate()$ instead of $H()$ and invoked $Simulate()$ instead of $H()$, thus never terminating. Every expert C programmer can see that this specifies infinite execution.

On this basis we know that the simulating halt decider H must stop simulating its input on the first line of $H_Hat()$.

This halt deciding principle overcomes the conventional halting problem proofs:

Every computation that never halts unless its simulation is aborted by its simulating halt decider is correctly decided as a computation that never halts.

When a chain of function calls specifies infinite recursion is broken by a simulating halt decider aborting the simulation of any one of these function calls, then the whole chain of function calls is correctly decided to specify a computation that does not halt.

This same reasoning applies to the computation: `H_Hat((u32)H_Hat)`; when `H_Hat()` invokes `H()` with its own machine address, this is the first invocation of an infinite chain of invocations. As the first element of an infinite chain of invocations where the third element of this chain is aborted the whole chain is understood to specify an infinite invocation sequence.

Execution trace of partial halt decider H deciding that its input `H_Hat()` would never halt because the execution trace of `H_Hat()` specifies an infinitely repeating pattern.

```

void H_Hat(u32 P)
{
    u32 Input_Halts = H(P, P);
    if (Input_Halts)
        HERE: goto HERE;
}

int main()
{
    u32 Input_Would_Halt6 = H((u32)H_Hat, (u32)H_Hat);
    Output("Input_Would_Halt6 = ", Input_Would_Halt6);
}

_H_Hat()
[0000b10] (01) 55          push ebp
[0000b11] (02) 8bec         mov ebp,esp
[0000b13] (01) 51          push ecx
[0000b14] (03) 8b4508      mov eax,[ebp+08]
[0000b17] (01) 50          push eax
[0000b18] (03) 8b4d08      mov ecx,[ebp+08]
[0000b1b] (01) 51          push ecx
[0000b1c] (05) e81ffeffff   call 00000940
[0000b21] (03) 83c408      add esp,+08
[0000b24] (03) 8945fc      mov [ebp-04],eax
[0000b27] (04) 837dfc00    cmp dword [ebp-04],+00
[0000b2b] (02) 7402         jz 0000b2f
[0000b2d] (02) ebfe         jmp 0000b2d
[0000b2f] (02) 8be5         mov esp,ebp
[0000b31] (01) 5d          pop ebp
[0000b32] (01) c3          ret
Size in bytes:(0035) [0000b32]

_main()
[0000b40] (01) 55          push ebp
[0000b41] (02) 8bec         mov ebp,esp
[0000b43] (01) 51          push ecx
[0000b44] (05) 68100b0000    push 0000b10
[0000b49] (05) 68100b0000    push 0000b10
[0000b4e] (05) e8edfdffff   call 00000940
[0000b53] (03) 83c408      add esp,+08
[0000b56] (03) 8945fc      mov [ebp-04],eax
[0000b59] (03) 8b45fc      mov eax,[ebp-04]
[0000b5c] (01) 50          push eax
[0000b5d] (05) 682b030000    push 000032b
[0000b62] (05) e8f9f7ffff   call 00000360
[0000b67] (03) 83c408      add esp,+08
[0000b6a] (02) 33c0         xor eax,eax
[0000b6c] (02) 8be5         mov esp,ebp
[0000b6e] (01) 5d          pop ebp
[0000b6f] (01) c3          ret
Size in bytes:(0048) [0000b6f]

```

Columns

- (1) Machine address of instruction
- (2) Machine address of top of stack
- (3) Value of top of stack after instruction executed
- (4) Number of bytes of machine code
- (5) Machine language bytes
- (6) Assembly language text

```

=====
... [0000b40] [00101533] [00000000] (01) 55          push ebp
... [0000b41] [00101533] [00000000] (02) 8bec       mov ebp,esp
... [0000b43] [0010152f] [00000000] (01) 51          push ecx
... [0000b44] [0010152b] [0000b10] (05) 68100b0000 push 0000b10
... [0000b49] [00101527] [0000b10] (05) 68100b0000 push 0000b10
... [0000b4e] [00101523] [0000b53] (05) e8edfdffff call 00000940
Begin Local Halt Decider Simulation at Machine Address:b10
... [0000b10] [002115d3] [002115d7] (01) 55          push ebp
... [0000b11] [002115d3] [002115d7] (02) 8bec       mov ebp,esp
... [0000b13] [002115cf] [002015a3] (01) 51          push ecx
... [0000b14] [002115cf] [002015a3] (03) 8b4508     mov eax,[ebp+08]
... [0000b17] [002115cb] [0000b10] (01) 50          push eax
... [0000b18] [002115cb] [0000b10] (03) 8b4d08     mov ecx,[ebp+08]
... [0000b1b] [002115c7] [0000b10] (01) 51          push ecx
... [0000b1c] [002115c3] [0000b21] (05) e81ffeffff call 00000940
... [0000b10] [0025bfffb] [0025bfff] (01) 55          push ebp
... [0000b11] [0025bfffb] [0025bfff] (02) 8bec       mov ebp,esp
... [0000b13] [0025bff7] [0024bfc3] (01) 51          push ecx
... [0000b14] [0025bff7] [0024bfc3] (03) 8b4508     mov eax,[ebp+08]
... [0000b17] [0025bff3] [0000b10] (01) 50          push eax
... [0000b18] [0025bff3] [0000b10] (03) 8b4d08     mov ecx,[ebp+08]
... [0000b1b] [0025bfe3] [0000b10] (01) 51          push ecx
... [0000b1c] [0025bfeb] [0000b21] (05) e81ffeffff call 00000940
Local Halt Decider: Infinite Recursion Detected Simulation Stopped
... [0000b53] [0010152f] [00000000] (03) 83c408     add esp,+08
... [0000b56] [0010152f] [00000000] (03) 8945fc     mov [ebp-04],eax
... [0000b59] [0010152f] [00000000] (03) 8b45fc     mov eax,[ebp-04]
... [0000b5c] [0010152b] [00000000] (01) 50          push eax
... [0000b5d] [00101527] [0000032b] (05) 682b030000 push 0000032b
... [0000b62] [00101527] [0000032b] (05) e8f9f7ffff call 00000360
Input_would_Halt6 = 0
... [0000b67] [0010152f] [00000000] (03) 83c408     add esp,+08
... [0000b6a] [0010152f] [00000000] (02) 33c0       xor eax,eax
... [0000b6c] [00101533] [00000000] (02) 8be5       mov esp,ebp
... [0000b6e] [00101537] [00100000] (01) 5d          pop ebp
... [0000b6f] [0010153b] [00000080] (01) c3         ret
Number_of_User_Instructions(33)
Number of Instructions Executed(26452)

```

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

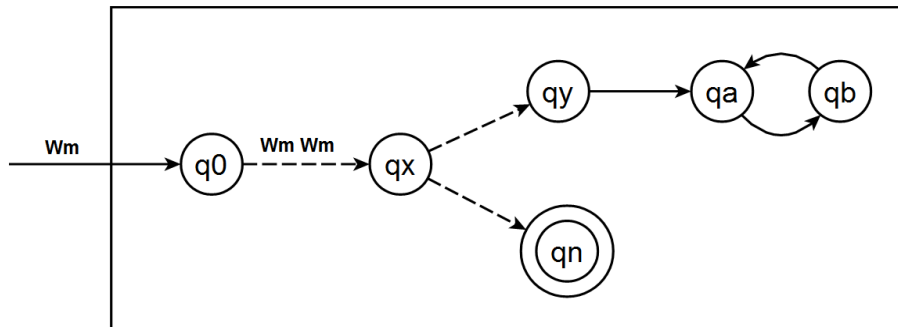


Figure 12.3 Turing Machine \hat{H}

$\hat{H}.q_0 wM \vdash^* \hat{H}.qx wM wM \vdash^* \hat{H}.qy \infty$
 $\hat{H}.q_0 wM \vdash^* \hat{H}.qx wM wM \vdash^* \hat{H}.qn$

The above is adapted from (Linz:1990:319).

It shows that Turing machine \hat{H} copies its input at (q_0) and begins executing an embedded copy of the original halt decider with this input at (qx).

The (qy) state indicates that the halt decider has determined that its input would halt. The (qn) state indicates the input would not halt. The appended (qa) and (qb) states cause \hat{H} to infinitely loop if the halt decider decides that its input would 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} . (Linz:1990:320)

If we imagine that the embedded halt decider at $\hat{H}.qx$ is simply a UTM then it seems quite easy to see that this would result in the following repeating pattern:

$\hat{H}.q_0$ copies its input then $\hat{H}.qx$ simulates this input with the copy then $\hat{H}.q_0$ copies its input then $\hat{H}.qx$ simulates this input with the copy then $\hat{H}.q_0$ copies its input then $\hat{H}.qx$ simulates this input with the copy then...

This is expressed in figure 12.4 as a cycle from qx to q_0 to qx .

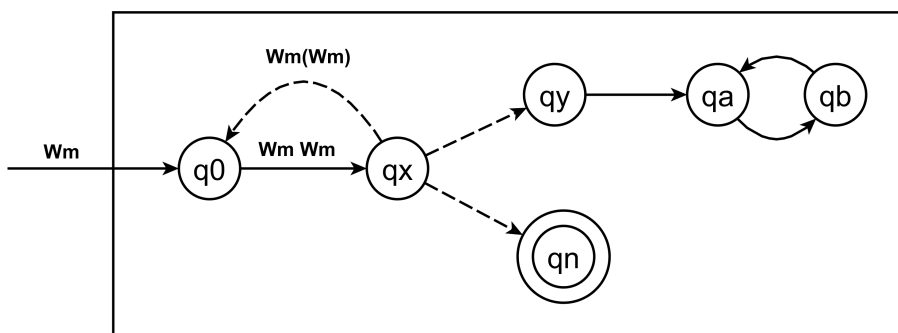


Figure 12.4 Turing Machine \hat{H}

When halting is defined as any computation that halts without ever having its simulation aborted by a simulating halt decider then we can see that (an at least partial) halt decider at state $\hat{H}.qx$ stops simulating its input and correctly transitions to its final $\hat{H}.qn$ state deciding not halting on its input in the computation: $\hat{H}(\hat{H})$.

When halting is defined as any computation that halts without ever having its simulation aborted then the fact that $\hat{H}(\hat{H})$ stops running does not indicate that it is a halting computation. The simulating halt decider would also force an infinite loop to stop running. The first invocation of infinite recursion is an element of the infinitely recursive chain.

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

Copyright 2021 PL Olcott