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 Ĥ.

When this simplified concrete example is fully understood then the exact same reasoning is applied to the actual Linz Ĥ 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 Ĥ (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()
{
  H_Hat((u32)H_Hat);
}


_H_Hat()
[00000af8](01)  55              push ebp
[00000af9](02)  8bec            mov ebp,esp
[00000afb](01)  51              push ecx
[00000afc](03)  8b4508          mov eax,[ebp+08]
[00000aff](01)  50              push eax
[00000b00](03)  8b4d08          mov ecx,[ebp+08]
[00000b03](01)  51              push ecx
[00000b04](05)  e81ffeffff      call 00000928
[00000b09](03)  83c408          add esp,+08
[00000b0c](03)  8945fc          mov [ebp-04],eax
[00000b0f](04)  837dfc00        cmp dword [ebp-04],+00
[00000b13](02)  7402            jz 00000b17
[00000b15](02)  ebfe            jmp 00000b15
[00000b17](02)  8be5            mov esp,ebp
[00000b19](01)  5d              pop ebp
[00000b1a](01)  c3              ret
Size in bytes:(0035) [00000b1a]

_main()
[00000b28](01)  55              push ebp
[00000b29](02)  8bec            mov ebp,esp
[00000b2b](05)  68f80a0000      push 00000af8
[00000b30](05)  e8c3ffffff      call 00000af8
[00000b35](03)  83c404          add esp,+04
[00000b38](02)  33c0            xor eax,eax
[00000b3a](01)  5d              pop ebp
[00000b3b](01)  c3              ret
Size in bytes:(0020) [00000b3b]
```

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

```
==================================
...[00000b28][001014cf][00000000](01)  55              push ebp
...[00000b29][001014cf][00000000](02)  8bec            mov ebp,esp
...[00000b2b][001014cb][00000af8](05)  68f80a0000      push 00000af8
...[00000b30][001014c7][00000b35](05)  e8c3ffffff      call 00000af8
...[00000af8][001014c3][001014cf](01)  55              push ebp
...[00000af9][001014c3][001014cf](02)  8bec            mov ebp,esp
...[00000afb][001014bf][00000000](01)  51              push ecx
...[00000afc][001014bf][00000000](03)  8b4508          mov eax,[ebp+08]
...[00000aff][001014bb][00000af8](01)  50              push eax
...[00000b00][001014bb][00000af8](03)  8b4d08          mov ecx,[ebp+08]
...[00000b03][001014b7][00000af8](01)  51              push ecx
...[00000b04][001014b3][00000b09](05)  e81ffeffff      call 00000928
Begin Local Halt Decider Simulation at Machine Address:af8
...[00000af8][0021156f][00211573](01)  55              push ebp
...[00000af9][0021156f][00211573](02)  8bec            mov ebp,esp
...[00000afb][0021156b][0020153f](01)  51              push ecx
...[00000afc][0021156b][0020153f](03)  8b4508          mov eax,[ebp+08]
...[00000aff][00211567][00000af8](01)  50              push eax
...[00000b00][00211567][00000af8](03)  8b4d08          mov ecx,[ebp+08]
...[00000b03][00211563][00000af8](01)  51              push ecx
...[00000b04][0021155f][00000b09](05)  e81ffeffff      call 00000928
...[00000af8][0025bf97][0025bf9b](01)  55              push ebp
...[00000af9][0025bf97][0025bf9b](02)  8bec            mov ebp,esp
...[00000afb][0025bf93][0024bf67](01)  51              push ecx
...[00000afc][0025bf93][0024bf67](03)  8b4508          mov eax,[ebp+08]
...[00000aff][0025bf8f][00000af8](01)  50              push eax
...[00000b00][0025bf8f][00000af8](03)  8b4d08          mov ecx,[ebp+08]
...[00000b03][0025bf8b][00000af8](01)  51              push ecx
...[00000b04][0025bf87][00000b09](05)  e81ffeffff      call 00000928
Local Halt Decider: Infinite Recursion Detected Simulation Stopped
```

H_Hat() only escapes infinitely nested simulation when H() aborts its simulation

```
...[00000b09][001014bf][00000000](03)  83c408          add esp,+08
...[00000b0c][001014bf][00000000](03)  8945fc          mov [ebp-04],eax
...[00000b0f][001014bf][00000000](04)  837dfc00        cmp dword [ebp-04],+00
...[00000b13][001014bf][00000000](02)  7402            jz 00000b17
...[00000b17][001014c3][001014cf](02)  8be5            mov esp,ebp
...[00000b19][001014c7][00000b35](01)  5d              pop ebp
...[00000b1a][001014cb][00000af8](01)  c3              ret
...[00000b35][001014cf][00000000](03)  83c404          add esp,+04
...[00000b38][001014cf][00000000](02)  33c0            xor eax,eax
...[00000b3a][001014d3][00100000](01)  5d              pop ebp
...[00000b3b][001014d7][00000098](01)  c3              ret
Number_of_User_Instructions(39)
Number of Instructions Executed(26459)
```

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

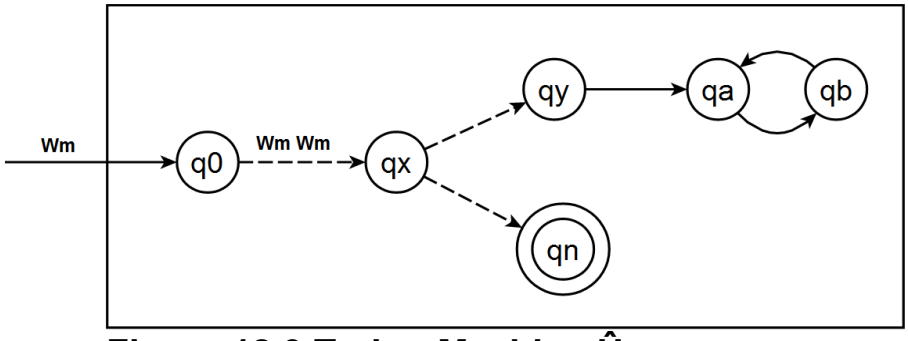

Figure 12.3 Turing Machine Ĥ

Ĥ.q0 wM ⊢* Ĥ.qx wM wM ⊢* Ĥ.qy ∞
Ĥ.q0 wM ⊢* Ĥ.qx wM wM ⊢* Ĥ.qn

The above is adapted from (Linz:1990:319).
It shows that Turing machine Ĥ copies its input at (q0) 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 Ĥ to infinitely loop if the halt decider decides that its input would halt.

> Now Ĥ is a Turing machine, so that it will have some description in Σ*, say ŵ. This string, in addition to being the description of Ĥ can also be used as input string. We can therefore legitimately ask what would happen if Ĥ is applied to ŵ. (Linz:1990:320)

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

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.

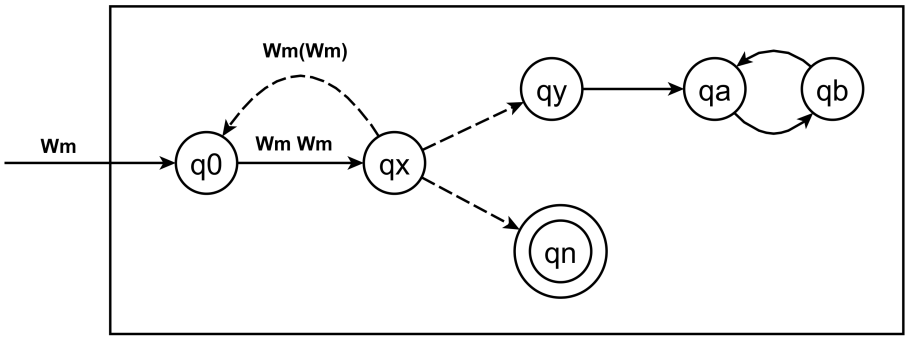

Figure 12.4 Turing Machine Ĥ

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 Ĥ.qx stops simulating its input and correctly transitions to its final Ĥ.qn state deciding not halting on its input in the computation: **Ĥ(⟨Ĥ⟩).**

When halting is defined as any computation that halts without ever having its simulation aborted then the fact that **Ĥ(⟨Ĥ⟩)** 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.