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 not halting on the simplified version of the Linz Ĥ. When this simplified concrete example is fully understood then the exact same reasoning can be applied to the actual Linz H correctly deciding not halting on its input.

The x86utm operating system was created so that the halting problem could be examined concretely in the high level language of C. x86utm 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 not halting.

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

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


void H_Hat2(u32 P)
{
  u32 Input_Halts = Simulate(P, P);
  if (Input_Halts)
    HERE: goto HERE;
}

int main()
{
  H_Hat2((u32)H_Hat2);
  H_Hat((u32)H_Hat);
}
```

Anyone that knows C programming very well will know that line 1 of main won't halt and 2 of main will only halt if simulating partial halt decider H stops simulating H_Hat(). A simulating halt decider that never stops simulating its input is simply a simulator on this input.

When we know that the UTM simulation of TM Description P on input I would never halt we know that the execution of TM P(I) would never halt.

On this basis we know that any computation that must have its simulation aborted to prevent its otherwise infinite execution is correctly rejected as non-halting.

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

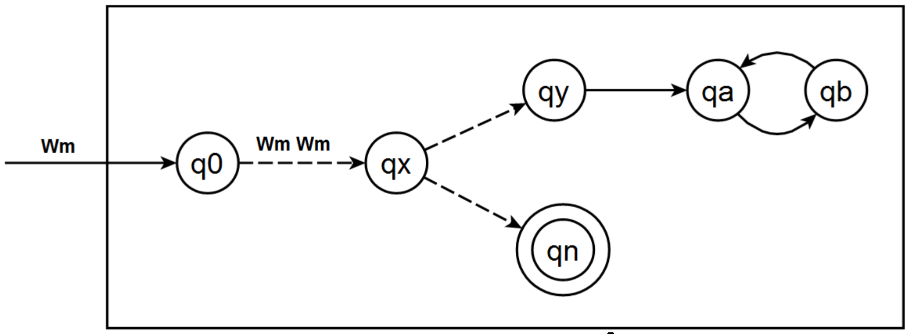

Figure 12.3 Turing Machine Ĥ

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.

The above definition specifies this execution trace:
It can be understood from the above specification that when the embedded halt decider at **Ĥ.qx** bases its halting decision on simulating its input, and it has **([Ĥ],[Ĥ])** as its input that:

- **Ĥ.q0** would copy its input and then **Ĥ.qx** would simulate its input with this copy then
- **Ĥ.q0** would copy its input and then **Ĥ.qx** would simulate its input with this copy then
- **Ĥ.q0** would copy its input and then **Ĥ.qx** would simulate its input with this copy...

unless and until the halt decider at **Ĥ.qx** stops simulating its input.

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 the 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.

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