Rebutting the Sipser Halting Problem Proof V2

A simulating halt decider correctly predicts what the behavior of its input would be if this simulated input never had its simulation aborted. It does this by correctly recognizing several non-halting behavior patterns in a finite number of steps of correct simulation.

When simulating halt decider H correctly predicts that directly executed D(D) would continue to run forever unless H aborts its simulation of D this directly applies to the halting theorem because H correctly determines:

> "from a description of an arbitrary computer program and an input, whether the program will finish running, or continue to run forever".
>
> For any program H that might determine whether programs halt, a "pathological" program D, called with some input, can pass its own source and its input to H and then specifically do the opposite of what H predicts D will do. https://en.wikipedia.org/wiki/Halting_problem

That (a) proves (b) is a tautology
(a) If simulating halt decider H correctly simulates its input D until H correctly determines that its simulated D would never stop running unless aborted then

(b) H can abort its simulation of D and correctly report that D specifies a non-halting sequence of configurations.

To make the details 100% concrete Sipser D and H are encoded as C
functions. The exact same rebuttal equally applies to Turing machines

```
int Sipser_D(int (*M)())
{
  int DoesHalt = H(M, M); // *Rejects when Sipser_D fails to accept*
  if (DoesHalt)
    return 0;
  return 1;
}

int main()
{
// *never stops running unless H aborts its simulation*
  Sipser_D(Sipser_D);
}
```

When Sipser_D calls H to simulate itself this comparable to calling H to call itself and can result in something like infinite recursion. Because there are no control flow instructions in Sipser_D to stop this the recursive simulation continues until H aborts it.

When the simulation of D is aborted this is comparable to a divide by zero error thus is not construed as D halting. **Entire system available:** https://github.com/plolcott/x86utm

This exact same principle works on all Turing machine based halting theorem proofs

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

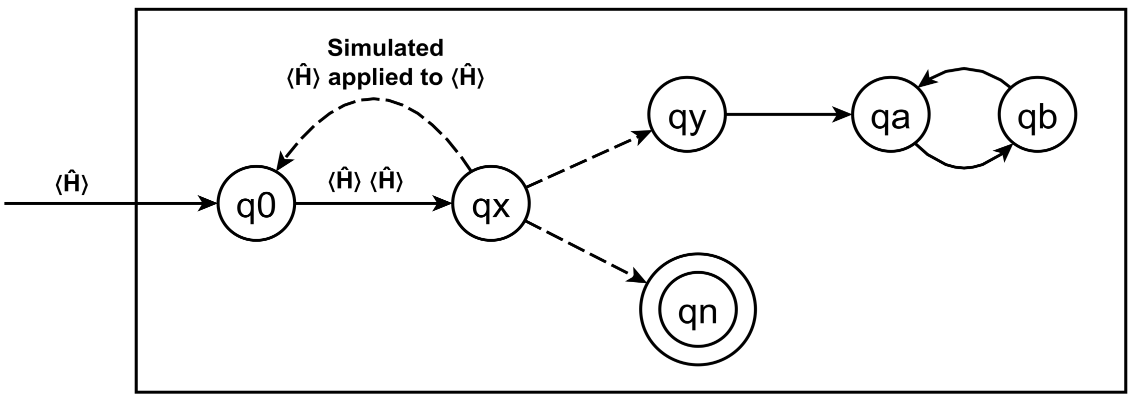
Applying a simulating halt decider to the Linz halting problem proof

A simulating halt decider correctly predicts what the behavior of its input would be if this simulated input never had its simulation aborted. It does this by correctly recognizing several non-halting behavior patterns in a finite number of steps of correct simulation.

The Linz text indicates that Ĥ is defined on the basis of prepending and appending states to the original Linz H, thus is named embedded_H.

(q0) is prepended to H to copy the ⟨Ĥ⟩ input of Ĥ. The transition from (qa) to (qb) is the conventional infinite loop appended to the (qy) accept state of **embedded_H.**
⊢* indicates an arbitrary number of moves such as: change_state / tape_head_action.
Ĥ is applied to its own machine description ⟨Ĥ⟩.

Ĥ.q0 ⟨Ĥ⟩ ⊢* embedded_H ⟨Ĥ⟩ ⟨Ĥ⟩ ⊢* Ĥ.qy ∞
Ĥ.q0 ⟨Ĥ⟩ ⊢* embedded_H ⟨Ĥ⟩ ⟨Ĥ⟩ ⊢* Ĥ.qn



When Ĥ is applied to ⟨Ĥ⟩
(q0) The input ⟨Ĥ⟩ is copied then transitions to (qx)
(qx) embedded_H is applied to ⟨Ĥ⟩ ⟨Ĥ⟩ (input and copy)
which simulates ⟨Ĥ⟩ ⟨Ĥ⟩ then transitions back to (q0) to repeat the process.

This process continues to repeat until embedded_H recognizes the repeating pattern and aborts its simulation of ⟨Ĥ⟩ ⟨Ĥ⟩.

embedded_H is correct to abort its simulation and transition to Ĥ.qn because it correctly predicts that Ĥ applied to ⟨Ĥ⟩ would never stop running unless embedded_H aborts its simulation of ⟨Ĥ⟩ ⟨Ĥ⟩.

Linz, Peter 1990. An Introduction to Formal Languages and Automata.
Lexington/Toronto: D. C. Heath and Company. (317-320)

A proof must connect an expression x of language L to its premises in L using only truth preserving operations or this "proof" diverges from correct reasoning. VALID DEDUCTION CORRECTED

True(L, x) applies only truth preserving operations to expression x of language L to connect x to expressions of L that are stipulated to be true. SOUND DEDUCTION CORRECTED

Within the above self-evidently correct definitions x cannot possibly be true in L and unprovable in L.