Termination Analyzer H is Not Fooled by Pathological Input D

When the halting problem is construed as requiring a correct yes/no answer to a contradictory question it cannot be solved. Any input D defined to do the opposite of whatever Boolean value that its termination analyzer H returns is a contradictory input relative to H. When H returns 1 for inputs that it determines do halt and returns 0 for inputs that either do not halt or do the opposite of whatever Boolean value that H returns then these pathological inputs are no longer contradictory and become decidable.

Can D correctly simulated by H terminate normally?
The x86utm operating system based on an open source x86 emulator. This system enables one C function to execute another C function in debug step mode. When H simulates D it creates a separate process context for D with its own memory, stack and virtual registers. H is able to simulate D simulating itself, thus the only limit to recursive simulations is RAM.

```
// The following is written in C
//
01 typedef int (*ptr)(); // pointer to int function
02 int H(ptr x, ptr y)   // uses x86 emulator to simulate its input
03
04 int D(ptr x)
05 {
06   int Halt_Status = H(x, x);
07   if (Halt_Status)
08     HERE: goto HERE;
09   return Halt_Status;
10 }
11
12 void main()
13 {
14   H(D,D);
15 }
```

Execution Trace
Line 14: main() invokes H(D,D);

keeps repeating (unless aborted)
Line 06: simulated D(D) invokes simulated H(D,D) that simulates D(D)

Simulation invariant:
D correctly simulated by H cannot possibly reach its own line 09.

H correctly determines that D correctly simulated by H cannot possibly terminate normally on the basis that H recognizes a dynamic behavior pattern equivalent to infinite recursion.
H outputs: **"H: Infinitely Recursive Simulation Detected Simulation Stopped"** indicating that D has defined a pathological (see above) relationship to H.

The x86utm operating system (includes several termination analyzers)
https://github.com/plolcott/x86utm

It compiles with the 2017 version of the Community Edition
https://visualstudio.microsoft.com/thank-you-downloading-visual-studio/?sku=Community&rel=15

Addendum applied to isomorphic Peter Linz Halting Problem Proof

A simulating (partial) halt decider correctly predicts whether or not its correctly simulated input can possibly reach its own final state and halt. It does this by correctly recognizing several non-halting behavior patterns in a finite number of steps of correct simulation. Inputs that do terminate are simply simulated until they complete.

When a simulating (partial) halt decider correctly simulates N steps of its input it derives the exact same N steps that a pure UTM would derive because it is itself a UTM with extra features.

My reviewers cannot show that any of the extra features added to the UTM change the behavior of the simulated input for the first N steps of simulation:

(a) Watching the behavior doesn't change it.
(b) Matching non-halting behavior patterns doesn't change it
(c) Even aborting the simulation after N steps doesn't change the first N steps.

Because of all this we can know that the first N steps of input D simulated by simulating (partial) halt decider H are the actual behavior that D presents to H for these same N steps.

computation that halts… "the Turing machine will halt whenever it enters a final state" (Linz:1990:234)

When we see (after N steps) that D correctly simulated by H cannot possibly reach its simulated final state in any finite number of steps of correct simulation then we have conclusive proof that D presents non-halting behavior to H.

A simulating (partial) halt decider must always stop its simulation and report non-halting when-so-ever it correctly detects that its correct simulation would never otherwise stop running. All halt deciders compute the mapping from their inputs to an accept or reject state on the basis of the actual behavior of this input.

When an input is defined to have a pathological relationship to its simulator this changes the behavior of this input. A simulating (partial) halt decider (with a pathological relationship) must report on this changed behavior to prevent its own infinite execution by aborting its simulation.

Summary of Linz Halting Problem Proof

The Linz halting problem proof constructs its counter-example input $\langle \hat{H} \rangle$ on the basis of prepending and appending states to the original Linz H, (assumed halt decider) thus is named embedded_H.

Original Linz Turing Machine H

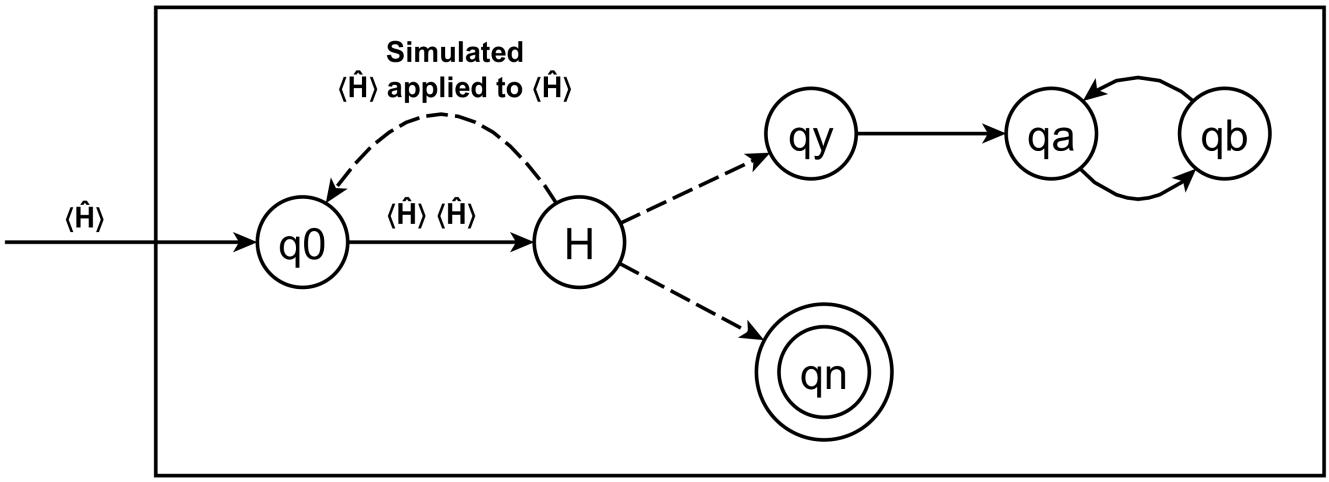
H.q0 ⟨M⟩ w ⊢* H.qy // Turing Machine description M and finite string w, accept state
H.q0 ⟨M⟩ w ⊢* Hqn // Turing Machine description M and finite string w, reject state

The Linz term "move" means a state transition and its corresponding tape head action {move_left, move_right, read, write}.

(q0) is prepended to H to copy the ⟨M⟩ 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.

Ĥ.q0 ⟨M⟩ ⊢* embedded_H ⟨M⟩ ⟨M⟩ ⊢* Ĥ.qy ∞ // see diagram below for (qa) and (qb)
Ĥ.q0 ⟨M⟩ ⊢* embedded_H ⟨M⟩ ⟨M⟩ ⊢* Ĥ.qn

Analysis of Linz Halting Problem Proof



When Ĥ is applied to ⟨Ĥ⟩
Ĥ.q0 ⟨Ĥ⟩ ⊢* embedded_H ⟨Ĥ⟩ ⟨Ĥ⟩ ⊢* Ĥ.qy ∞
Ĥ.q0 ⟨Ĥ⟩ ⊢* embedded_H ⟨Ĥ⟩ ⟨Ĥ⟩ ⊢* Ĥ.qn

computation that halts… "the Turing machine will halt whenever it enters a final state" (Linz:1990:234)

Non-halting behavior patterns can be matched in N steps
⟨Ĥ⟩ Halting is reaching its simulated final state of ⟨Ĥ.qn⟩ in a finite number of steps

N steps of ⟨Ĥ⟩ correctly simulated by embedded_H are the actual behavior of this input:
(a) Ĥ.q0 The input ⟨Ĥ⟩ is copied then transitions to embedded_H
(b) embedded_H is applied to ⟨Ĥ⟩ ⟨Ĥ⟩ (input and copy) which simulates ⟨Ĥ⟩ applied to ⟨Ĥ⟩
(c) **which begins at its own simulated ⟨Ĥ.q0⟩ to repeat the process**

Simulation invariant: ⟨Ĥ⟩ correctly simulated by embedded_H never reaches its own simulated final state of ⟨Ĥ.qn⟩. This also means that Ĥ has defined a pathological relationship to embedded_H.

Therefore when embedded_H aborts the simulation of its input and transitions to its own final state of Ĥ.qn it is merely reporting this verified fact.

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