Halting problem undecidability and infinitely nested simulation

There seems to be a huge gap in the reasoning of the halting problem proofs. All of the conventional halting problem proofs simply assume that halt decider H must return a correct halt status of its input P to its input.

None of these proofs consider the possibility that a simulating halt decider would be required to abort the simulation of its input before ever returning any value to this input. If the input to a simulating halt decider specifies infinitely nested simulation then the halt decider must abort its simulation of this input.

When a simulating partial halt decider H is applied to a simplified concrete example P of the Peter Linz Ĥ template the details of this process show that P(P) is a computation that never halts unless it is aborted at some point. This same reasoning is then applied to the actual Ĥ(⟨Ĥ⟩) computation of the Peter Linz proof.

In the concrete example shown below a simulating halt decider is based on a x86 emulator. In the Turing machine model it is based on a Universal Turing Machine (UTM). In both of these cases the input is simulated one instruction at a time.

Then the stored execution trace is compared to patterns of behavior that never halt. Simulating halt deciders continue to act only as simulators until the execution trace of their input matches a non-halting behavior pattern.

The only two patterns that are examined here are (a) Infinite loops (b) Infinite recursion / Infinitely nested simulation. When a simulating halt decider matches one of these patterns it aborts the simulation of its input and reports that its input does not halt.

Because a simulating halt decider must always abort the simulation of every input that never halts its halt deciding criteria must be adapted: **Does the input halt on its input?** must become: **Does the input halt without having its simulation aborted?**
This change is required because every input to a simulating halt decider either halts on its own or halts because its simulation has been aborted.

The standard pseudo-code halting problem template "proved" that the halting problem could never be solved on the basis that neither value of true (halting) nor false (not halting) could be correctly returned to the confounding input.

```
procedure compute_g(i):
  if f(i, i) == 0 then
    return 0
  else
    loop forever    // (Wikipedia:Halting Problem)
```

This problem is overcome on the basis that a simulating halt decider would abort the simulation of its input before ever returning any value to this input. It aborts the simulation of its input on the basis that its input specifies what is essentially infinite recursion (infinitely nested simulation) to any simulating halt decider.

The x86utm operating system was created so that the halting problem could be examined concretely in the high level language of C and x86. When examining the halting problem this way every detail can be explicitly specified. UTM tape elements are 32-bit unsigned integers.

```
// Simplified Linz Ĥ (Linz:1990:319)
void P(u32 x)
{
  u32 Input_Halts = H(x, x);
  if (Input_Halts)
    HERE: goto HERE;
}

int main()
{
  u32 Input_Halts = H((u32)P, (u32)P);
  Output("Input_Halts = ", Input_Halts);
}
```

H analyzes the (currently updated) stored execution trace of its x86 emulation of P(P) after it simulates each instruction of input (P, P). 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.

A simulating halt decider must abort the simulation of every input that never halts. For H to recognize the infinitely repeating pattern of P it only needs to see that same thing that humans see when they examine the x86 execution trace of the simulation of P. All of these details including the complete x86 execution trace of P(P) is provided below.

To anchor these ideas in a very simple concrete example we show how H decides that an infinite loop never halts.

Simulating partial halt decider H correctly decides that Infinite_Loop() never halts

```
void Infinite_Loop()
{
  HERE: goto HERE;
}

int main()
{
  u32 Input_Would_Halt2 = H((u32)Infinite_Loop, (u32)Infinite_Loop);
  Output("Input_Would_Halt2 = ", Input_Would_Halt2);
}

_Infinite_Loop()
[00000ab0](01)  55              push ebp
[00000ab1](02)  8bec            mov ebp,esp
[00000ab3](02)  ebfe            jmp 00000ab3
[00000ab5](01)  5d              pop ebp
[00000ab6](01)  c3              ret
Size in bytes:(0007) [00000ab6]
```

```
_main()
[00000c00](01)  55              push ebp
[00000c01](02)  8bec            mov ebp,esp
[00000c03](01)  51              push ecx
[00000c04](05)  68b00a0000      push 00000ab0
[00000c09](05)  68b00a0000      push 00000ab0
[00000c0e](05)  e84dfdffff      call 00000960
[00000c13](03)  83c408          add esp,+08
[00000c16](03)  8945fc          mov [ebp-04],eax
[00000c19](03)  8b45fc          mov eax,[ebp-04]
[00000c1c](01)  50              push eax
[00000c1d](05)  684b030000      push 0000034b
[00000c22](05)  e859f7ffff      call 00000380
[00000c27](03)  83c408          add esp,+08
[00000c2a](02)  33c0            xor eax,eax
[00000c2c](02)  8be5            mov esp,ebp
[00000c2e](01)  5d              pop ebp
[00000c2f](01)  c3              ret
Size in bytes:(0048) [00000c2f]
===============================
...[00000c00][00101693][00000000](01)  55              push ebp
...[00000c01][00101693][00000000](02)  8bec            mov ebp,esp
...[00000c03][0010168f][00000000](01)  51              push ecx
...[00000c04][0010168b][00000ab0](05)  68b00a0000      push 00000ab0
...[00000c09][00101687][00000ab0](05)  68b00a0000      push 00000ab0
...[00000c0e][00101683][00000c13](05)  e84dfdffff      call 00000960

Begin Local Halt Decider Simulation at Machine Address:ab0
...[00000ab0][00211733][00211737](01)  55              push ebp
...[00000ab1][00211733][00211737](02)  8bec            mov ebp,esp
...[00000ab3][00211733][00211737](02)  ebfe            jmp 00000ab3
...[00000ab3][00211733][00211737](02)  ebfe            jmp 00000ab3
Local Halt Decider: Infinite Loop Detected Simulation Stopped

...[00000c13][0010168f][00000000](03)  83c408          add esp,+08
...[00000c16][0010168f][00000000](03)  8945fc          mov [ebp-04],eax
...[00000c19][0010168f][00000000](03)  8b45fc          mov eax,[ebp-04]
...[00000c1c][0010168b][00000000](01)  50              push eax
...[00000c1d][00101687][0000034b](05)  684b030000      push 0000034b
...[00000c22][00101687][0000034b](05)  e859f7ffff      call 00000380
Input_Would_Halt2 = 0
...[00000c27][0010168f][00000000](03)  83c408          add esp,+08
...[00000c2a][0010168f][00000000](02)  33c0            xor eax,eax
...[00000c2c][00101693][00000000](02)  8be5            mov esp,ebp
...[00000c2e][00101697][00100000](01)  5d              pop ebp
...[00000c2f][0010169b][00000050](01)  c3              ret
Number_of_User_Instructions(21)
Number of Instructions Executed(640)
```

Simulating partial halt decider H decides that Infinite_Recursion() never halts

```
void Infinite_Recursion(u32 N)
{
  Infinite_Recursion(N);
}

int main()
{
   u32 Input_Halts = H((u32)Infinite_Recursion, 3);
   Output("Input_Halts = ", Input_Halts);
}
```

```
_Infinite_Recursion()
[00000ac6](01)  55                push ebp
[00000ac7](02)  8bec              mov ebp,esp
[00000ac9](03)  8b4508            mov eax,[ebp+08]
[00000acc](01)  50                push eax
[00000acd](05)  e8f4ffffff        call 00000ac6
[00000ad2](03)  83c404            add esp,+04
[00000ad5](01)  5d                pop ebp
[00000ad6](01)  c3                ret
Size in bytes:(0017) [00000ad6]

_main()
[00000c46](01)  55                push ebp
[00000c47](02)  8bec              mov ebp,esp
[00000c49](01)  51                push ecx
[00000c4a](02)  6a03              push +03
[00000c4c](05)  68c60a0000        push 00000ac6
[00000c51](05)  e810fdffff        call 00000966
[00000c56](03)  83c408            add esp,+08
[00000c59](03)  8945fc            mov [ebp-04],eax
[00000c5c](03)  8b45fc            mov eax,[ebp-04]
[00000c5f](01)  50                push eax
[00000c60](05)  6857030000        push 00000357
[00000c65](05)  e81cf7ffff        call 00000386
[00000c6a](03)  83c408            add esp,+08
[00000c6d](02)  33c0              xor eax,eax
[00000c6f](02)  8be5              mov esp,ebp
[00000c71](01)  5d                pop ebp
[00000c72](01)  c3                ret
Size in bytes:(0045) [00000c72]
```

Columns
(1) Machine address of instruction
(2) Machine address of top of stack
(3) Value of top of stack after instruction executed
(4) Machine language bytes
(5) Assembly language text

```
===============================
...[00000c46][001016fa][00000000](01)  55              push ebp
...[00000c47][001016fa][00000000](02)  8bec            mov ebp,esp
...[00000c49][001016f6][00000000](01)  51              push ecx
...[00000c4a][001016f2][00000003](02)  6a03            push +03
...[00000c4c][001016ee][00000ac6](05)  68c60a0000      push 00000ac6
...[00000c51][001016ea][00000c56](05)  e810fdffff      call 00000966

Begin Local Halt Decider Simulation at Machine Address:ac6
...[00000ac6][0021179a][0021179e](01)  55              push ebp
...[00000ac7][0021179a][0021179e](02)  8bec            mov ebp,esp
...[00000ac9][0021179a][0021179e](03)  8b4508          mov eax,[ebp+08]
...[00000acc][00211796][00000003](01)  50              push eax
...[00000acd][00211792][00000ad2](05)  e8f4ffffff      call 00000ac6
...[00000ac6][0021178e][0021179a](01)  55              push ebp
...[00000ac7][0021178e][0021179a](02)  8bec            mov ebp,esp
...[00000ac9][0021178e][0021179a](03)  8b4508          mov eax,[ebp+08]
...[00000acc][0021178a][00000003](01)  50              push eax
...[00000acd][00211786][00000ad2](05)  e8f4ffffff      call 00000ac6
Local Halt Decider: Infinite Recursion Detected Simulation Stopped
```

_Infinite_Recursion() calls itself recursively with the same input. It has no escape from this infinite recursion. H recognizes this infinite behavior pattern, aborts its simulation of _Infinite_Recursion() and reports that this input never halts.

```
...[00000c56][001016f6][00000000](03)  83c408          add esp,+08
...[00000c59][001016f6][00000000](03)  8945fc          mov [ebp-04],eax
...[00000c5c][001016f6][00000000](03)  8b45fc          mov eax,[ebp-04]
...[00000c5f][001016f2][00000000](01)  50              push eax
...[00000c60][001016ee][00000357](05)  6857030000      push 00000357
...[00000c65][001016ee][00000357](05)  e81cf7ffff      call 00000386
Input_Halts = 0
...[00000c6a][001016f6][00000000](03)  83c408          add esp,+08
...[00000c6d][001016f6][00000000](02)  33c0            xor eax,eax
...[00000c6f][001016fa][00000000](02)  8be5            mov esp,ebp
...[00000c71][001016fe][00100000](01)  5d              pop ebp
...[00000c72][00101702][00000068](01)  c3              ret
Number_of_User_Instructions(27)
Number of Instructions Executed(1240)
```

Simulating partial halt decider H correctly decides that P(P) never halts (V1)

When a simulating halt decider only simulates its input until it detects that its input exhibits non-halting behavior then we can know that this simulating halt decider has no effect what-so-ever on the behavior of this input.

This also means that while a simulating halt decider is examining the behavior of its input it can safely ignore its own behavior. When this simulating halt decider does detect an infinite execution behavior pattern then it can correctly stop simulating its input and report that its input does not halt.

```
// Simplified Linz Ĥ (Linz:1990:319)
void P(u32 x)
{
  u32 Input_Halts = H(x, x);
  if (Input_Halts)
    HERE: goto HERE;
}


int main()
{
  u32 Input_Halts = H((u32)P, (u32)P);
  Output("Input_Halts = ", Input_Halts);
}

_P()
[00000b1a](01)  55              push ebp
[00000b1b](02)  8bec            mov ebp,esp
[00000b1d](01)  51              push ecx
[00000b1e](03)  8b4508          mov eax,[ebp+08]
[00000b21](01)  50              push eax        // 2nd Param
[00000b22](03)  8b4d08          mov ecx,[ebp+08]
[00000b25](01)  51              push ecx        // 1st Param
[00000b26](05)  e81ffeffff      call 0000094a   // call H
[00000b2b](03)  83c408          add esp,+08
[00000b2e](03)  8945fc          mov [ebp-04],eax
[00000b31](04)  837dfc00        cmp dword [ebp-04],+00
[00000b35](02)  7402            jz 00000b39
[00000b37](02)  ebfe            jmp 00000b37
[00000b39](02)  8be5            mov esp,ebp
[00000b3b](01)  5d              pop ebp
[00000b3c](01)  c3              ret
Size in bytes:(0035) [00000b3c]
```

```
_main()
[00000bda](01)  55              push ebp
[00000bdb](02)  8bec            mov ebp,esp
[00000bdd](01)  51              push ecx
[00000bde](05)  681a0b0000      push 00000b1a  // push address of P
[00000be3](05)  681a0b0000      push 00000b1a  // push address of P
[00000be8](05)  e85dfdffff      call 0000094a  // call H
[00000bed](03)  83c408          add esp,+08
[00000bf0](03)  8945fc          mov [ebp-04],eax
[00000bf3](03)  8b45fc          mov eax,[ebp-04]
[00000bf6](01)  50              push eax
[00000bf7](05)  683b030000      push 0000033b
[00000bfc](05)  e869f7ffff      call 0000036a
[00000c01](03)  83c408          add esp,+08
[00000c04](02)  33c0            xor eax,eax
[00000c06](02)  8be5            mov esp,ebp
[00000c08](01)  5d              pop ebp
[00000c09](01)  c3              ret
Size in bytes:(0048) [00000c09]
```

Columns
(1) Machine address of instruction
(2) Machine address of top of stack
(3) Value of top of stack after instruction executed
(4) Machine language bytes
(5) Assembly language text

```
===============================
...[00000bda][00101647][00000000](01)  55              push ebp
...[00000bdb][00101647][00000000](02)  8bec            mov ebp,esp
...[00000bdd][00101643][00000000](01)  51              push ecx
...[00000bde][0010163f][00000b1a](05)  681a0b0000      push 00000b1a // push P
...[00000be3][0010163b][00000b1a](05)  681a0b0000      push 00000b1a // push P
...[00000be8][00101637][00000bed](05)  e85dfdffff      call 0000094a // call H

Begin Local Halt Decider Simulation at Machine Address:b1a
...[00000b1a][002116e7][002116eb](01)  55              push ebp
...[00000b1b][002116e7][002116eb](02)  8bec            mov ebp,esp
...[00000b1d][002116e3][002016b7](01)  51              push ecx
...[00000b1e][002116e3][002016b7](03)  8b4508          mov eax,[ebp+08]
...[00000b21][002116df][00000b1a](01)  50              push eax      // push P
...[00000b22][002116df][00000b1a](03)  8b4d08          mov ecx,[ebp+08]
...[00000b25][002116db][00000b1a](01)  51              push ecx      // push P
...[00000b26][002116d7][00000b2b](05)  e81ffeffff      call 0000094a // call H
...[00000b1a][0025c10f][0025c113](01)  55              push ebp
...[00000b1b][0025c10f][0025c113](02)  8bec            mov ebp,esp
...[00000b1d][0025c10b][0024c0df](01)  51              push ecx
...[00000b1e][0025c10b][0024c0df](03)  8b4508          mov eax,[ebp+08]
...[00000b21][0025c107][00000b1a](01)  50              push eax      // push P
...[00000b22][0025c107][00000b1a](03)  8b4d08          mov ecx,[ebp+08]
...[00000b25][0025c103][00000b1a](01)  51              push ecx      // push P
...[00000b26][0025c0ff][00000b2b](05)  e81ffeffff      call 0000094a // call H
Local Halt Decider: Infinite Recursion Detected Simulation Stopped
```

In the above 16 instructions of the simulation of P(P) we can see that the first 8 instructions of P are repeated. The end of this sequence of 8 instructions P calls H with its own machine address as the parameters to H: H(P,P). Because H only examines the behavior of its inputs and ignores its own behavior when H(P,P) is called we only see the first instruction of P being simulated.

Anyone knowing the x86 language well enough can see that none of these 8 simulated instructions of P have any escape from their infinitely repeating behavior pattern. When H recognizes this infinitely repeating pattern it aborts its simulation of P(P) and reports that its input: (P,P) would never halt on its input.

```
...[00000bed][00101643][00000000](03)  83c408          add esp,+08
...[00000bf0][00101643][00000000](03)  8945fc          mov [ebp-04],eax
...[00000bf3][00101643][00000000](03)  8b45fc          mov eax,[ebp-04]
...[00000bf6][0010163f][00000000](01)  50              push eax
...[00000bf7][0010163b][0000033b](05)  683b030000      push 0000033b
...[00000bfc][0010163b][0000033b](05)  e869f7ffff      call 0000036a
Input_Halts = 0
...[00000c01][00101643][00000000](03)  83c408          add esp,+08
...[00000c04][00101643][00000000](02)  33c0            xor eax,eax
...[00000c06][00101647][00000000](02)  8be5            mov esp,ebp
...[00000c08][0010164b][00100000](01)  5d              pop ebp
...[00000c09][0010164f][00000080](01)  c3              ret
Number_of_User_Instructions(33)
Number of Instructions Executed(26452)
```

This is the sound deductive inference (proof) that H(P,P)==0 is correct.

Premise(1) (axiom) Every computation that never halts unless its simulation is aborted is a computation that never halts. This verified as true on the basis of the meaning of its words.

Premise(2) (verified fact) The simulation of the input to H(P,P) never halts without being aborted is a verified fact on the basis of its x86 execution trace. (shown below).

When the simulator determines whether or not it must abort the simulation of its input based on the behavior of its input the simulator only acts as an x86 emulator thus has no effect on the behavior of its input. This allows the simulator to always ignore its own behavior.

Conclusion(3) From the above true premises it necessarily follows that simulating halt decider H correctly reports that its input: (P,P) never halts.

Simulating partial halt decider H correctly decides that P(P) never halts (V2)

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

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

```
_P()
[00000b25](01)  55              push ebp
[00000b26](02)  8bec            mov ebp,esp
[00000b28](01)  51              push ecx
[00000b29](03)  8b4508          mov eax,[ebp+08]
[00000b2c](01)  50              push eax
[00000b2d](03)  8b4d08          mov ecx,[ebp+08]
[00000b30](01)  51              push ecx
[00000b31](05)  e81ffeffff      call 00000955
[00000b36](03)  83c408          add esp,+08
[00000b39](03)  8945fc          mov [ebp-04],eax
[00000b3c](04)  837dfc00        cmp dword [ebp-04],+00
[00000b40](02)  7402            jz 00000b44
[00000b42](02)  ebfe            jmp 00000b42
[00000b44](02)  8be5            mov esp,ebp
[00000b46](01)  5d              pop ebp
[00000b47](01)  c3              ret
Size in bytes:(0035) [00000b47]

_main()
[00000c05](01)  55              push ebp
[00000c06](02)  8bec            mov ebp,esp
[00000c08](05)  68250b0000      push 00000b25
[00000c0d](05)  e813ffffff      call 00000b25
[00000c12](03)  83c404          add esp,+04
[00000c15](02)  33c0            xor eax,eax
[00000c17](01)  5d              pop ebp
[00000c18](01)  c3              ret
Size in bytes:(0020) [00000c18]
```

Columns

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

```
===============================
...[00000c05][0010165e][00000000](01)  55              push ebp
...[00000c06][0010165e][00000000](02)  8bec            mov ebp,esp
...[00000c08][0010165a][00000b25](05)  68250b0000      push 00000b25
...[00000c0d][00101656][00000c12](05)  e813ffffff      call 00000b25
...[00000b25][00101652][0010165e](01)  55              push ebp
...[00000b26][00101652][0010165e](02)  8bec            mov ebp,esp
...[00000b28][0010164e][00000000](01)  51              push ecx
...[00000b29][0010164e][00000000](03)  8b4508          mov eax,[ebp+08]
...[00000b2c][0010164a][00000b25](01)  50              push eax
...[00000b2d][0010164a][00000b25](03)  8b4d08          mov ecx,[ebp+08]
...[00000b30][00101646][00000b25](01)  51              push ecx
...[00000b31][00101642][00000b36](05)  e81ffeffff      call 00000955
```

```
Begin Local Halt Decider Simulation at Machine Address:b25
...[00000b25][002116fe][00211702](01)  55              push ebp
...[00000b26][002116fe][00211702](02)  8bec            mov ebp,esp
...[00000b28][002116fa][002016ce](01)  51              push ecx
...[00000b29][002116fa][002016ce](03)  8b4508          mov eax,[ebp+08]
...[00000b2c][002116f6][00000b25](01)  50              push eax
...[00000b2d][002116f6][00000b25](03)  8b4d08          mov ecx,[ebp+08]
...[00000b30][002116f2][00000b25](01)  51              push ecx
...[00000b31][002116ee][00000b36](05)  e81ffeffff      call 00000955
...[00000b25][0025c126][0025c12a](01)  55              push ebp
...[00000b26][0025c126][0025c12a](02)  8bec            mov ebp,esp
...[00000b28][0025c122][0024c0f6](01)  51              push ecx
...[00000b29][0025c122][0024c0f6](03)  8b4508          mov eax,[ebp+08]
...[00000b2c][0025c11e][00000b25](01)  50              push eax
...[00000b2d][0025c11e][00000b25](03)  8b4d08          mov ecx,[ebp+08]
...[00000b30][0025c11a][00000b25](01)  51              push ecx
...[00000b31][0025c116][00000b36](05)  e81ffeffff      call 00000955
Local Halt Decider: Infinite Recursion Detected Simulation Stopped

...[00000b36][0010164e][00000000](03)  83c408          add esp,+08
...[00000b39][0010164e][00000000](03)  8945fc          mov [ebp-04],eax
...[00000b3c][0010164e][00000000](04)  837dfc00        cmp dword [ebp-04],+00
...[00000b40][0010164e][00000000](02)  7402            jz 00000b44
...[00000b44][00101652][0010165e](02)  8be5            mov esp,ebp
...[00000b46][00101656][00000c12](01)  5d              pop ebp
...[00000b47][0010165a][00000b25](01)  c3              ret
...[00000c12][0010165e][00000000](03)  83c404          add esp,+04
...[00000c15][0010165e][00000000](02)  33c0            xor eax,eax
...[00000c17][00101662][00100000](01)  5d              pop ebp
...[00000c18][00101666][00000098](01)  c3              ret
Number_of_User_Instructions(39)
Number of Instructions Executed(26459)
```

In the computation **int main() { P(P); }** when no P ever halts unless some H aborts some P this proves beyond all possible doubt that P(P) specifies an infinitely recursive chain of invocations.

The computation **int main() { P(P); }** calls H(P,P) which is the first invocation of an infinite chain of invocations. Whenever P calls H(P,P) H must abort its simulation of P.

It is common knowledge that when any invocation of an infinite sequence of invocations (such as infinite recursion or infinitely nested simulation) is terminated then the entire sequence halts at the point of termination.

In the computation **int main() { P(P); }** the third element of the infinite chain of invocations is terminated. The only reason that any P ever halts is that some H aborted some P. This proves (axiomatically) that P(P) really does specify an infinite invocation chain.

(Axiom) Every computation that never halts unless it is aborted at some point is a computation that never halts. This verified as true on the basis of the meaning of its words.

Infinite recursion detection criteria:
If the execution trace of function X() called by function Y() shows:
(1) Function X() is called twice in sequence from the same machine address of Y().
(2) With the same parameters to X().
(3) With no conditional branch or indexed jump instructions in Y().
(4) With no function call returns from X().
then the function call from Y() to X() is infinitely recursive unless X() stops it.

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

The following simplifies the syntax for the definition of the Linz Turing machine Ĥ, it is now a single machine with a single start state. The halt decider is embedded at state Ĥ.qx.

Ĥ.q0 wM ⊢* Ĥ.qx wM wM ⊢* Ĥ.qy ∞
if M applied to wM halts, and

Ĥ.q0 wM ⊢* Ĥ.qx wM wM ⊢* Ĥ.qn
if M applied to wM does not halt

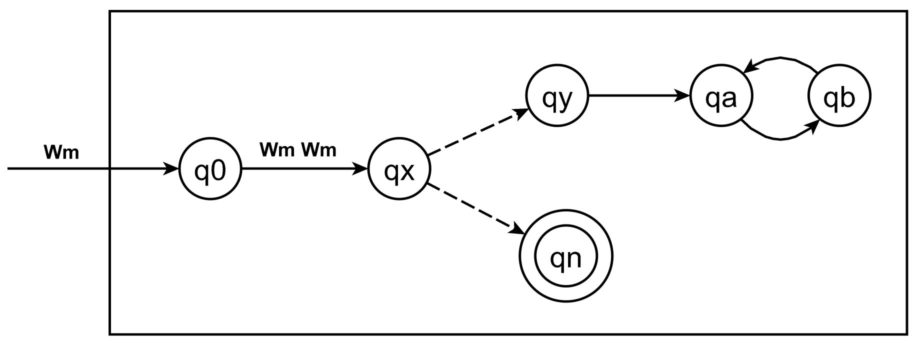


Figure 12.3 Turing Machine Ĥ

To provide a sketch of the idea of how a simulating halt decider would analyze the Peter Linz Ĥ applied to its own Turing machine description we start by examining the behavior of an ordinary UTM.

When we hypothesize that the halt decider embedded in Ĥ is simply a UTM then it seems that when the Peter Linz Ĥ is applied to its own Turing machine description ⟨Ĥ⟩ this specifies a computation that never halts.

$\hat{H}_0$.q0 copies its input ⟨$\hat{H}_1$⟩ to ⟨$\hat{H}_x$⟩ then $\hat{H}_0$.qx simulates this input with the copy then
$\hat{H}_1$.q0 copies its input ⟨$\hat{H}_2$⟩ to ⟨$\hat{H}_y$⟩ then $\hat{H}_1$.qx simulates this input with the copy then
$\hat{H}_2$.q0 copies its input ⟨$\hat{H}_3$⟩ to ⟨$\hat{H}_z$⟩ then $\hat{H}_2$.qx simulates this input with the copy then ...

This is expressed in figure 12.4 as a cycle from qx to q0 to qx.

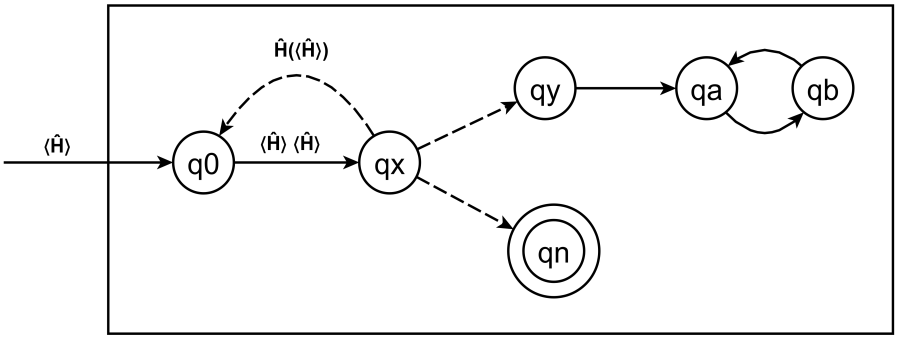


Figure 12.4 Turing Machine Ĥ applied to ⟨Ĥ⟩ input

Within the hypothesis that the internal halt decider embedded within Ĥ simulates its input Ĥ applied to its own Turing machine description ⟨Ĥ⟩ derives infinitely nested simulation, unless this simulation is aborted.

Self-Evident-Truth (premise[1])
Every computation that never halts unless its simulation is aborted is a computation that never halts.

Self-Evident-Truth (premise[2])
The ⟨Ĥ⟩ ⟨Ĥ⟩ input to the embedded halt decider at Ĥ.qx is a computation that never halts unless its simulation is aborted.

∴ **Sound Deductive Conclusion**
The embedded simulating halt decider at Ĥ.qx correctly decides its input: ⟨Ĥ⟩ ⟨Ĥ⟩ is a computation that never halts.

Ĥ.q0 ⟨Ĥ⟩ specifies an infinite chain of invocations that is terminated at its third invocation. The first invocation of Ĥ.qx ⟨Ĥ⟩, ⟨Ĥ⟩ is the first element of an infinite chain of invocations.

It is common knowledge that when any invocation of an infinite chain of invocations is terminated that the whole chain terminates. That the first element of this infinite chain terminates after its third element has been terminated does not entail that this first element is an actual terminating computation.

For the first element to be an actual terminating computation it must terminate without any of the elements of the infinite chain of invocations being terminated.

Linz, Peter 1990. An Introduction to Formal Languages and Automata. Lexington/Toronto: D. C. Heath and Company. (318-320)

Theorem 12.1

There does not exist any Turing machine H that behaves as required by Definition 12.1. The halting problem is therefore undecidable.

Proof: We assume the contrary, namely that there exists an algorithm, and consequently some Turing machine H, that solves the halting problem. The input to H will be the description (encoded in some form) of M, say w_M, as well as the input w. The requirement is then that, given any (w_M, w), the Turing machine H will halt with either a yes or no answer. We achieve this by asking that H halt in one of two corresponding final states, say, q_y or q_n. The situation can be visualized by a block diagram like Figure 12.1. The intent of this diagram is to indicate that, if M is started in state q_0 with input (w_M, w), it will eventually halt in state q_y or q_n. As required by Definition 12.1, we want H to operate according to the following rules:

$$q_0 w_M w \overset{*}{\vdash}_H x_1 q_y x_2,$$

if M applied to w halts, and

$$q_0 w_M w \overset{*}{\vdash}_H y_1 q_n y_2,$$

if M applied to w does not halt.

Figure 12.1

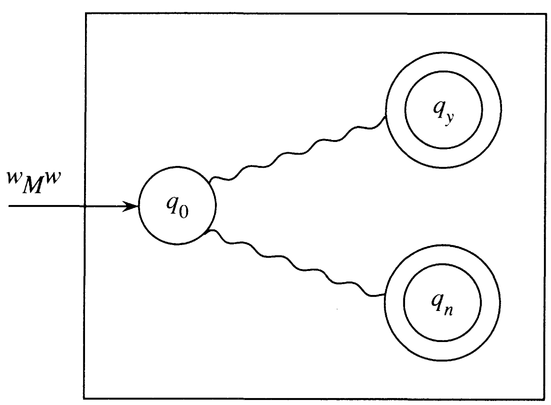
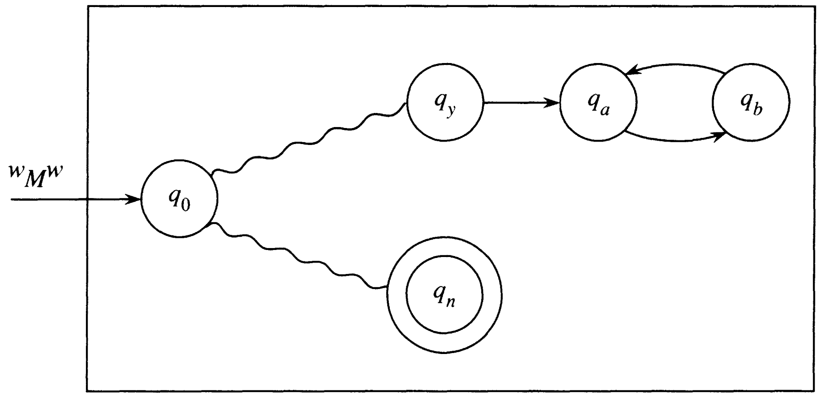


Figure 12.2



Next, we modify H to produce a Turing machine H' with the structure shown in Figure 12.2. With the added states in Figure 12.2 we want to convey that the transitions between state q_y and the new states q_a and q_b are to be made, regardless of the tape symbol, in such a way that the tape remains unchanged. The way this is done is straightforward. Comparing H and H' we see that, in situations where H reaches q_y and halts, the modified machine H' will enter an infinite loop. Formally, the action of H' is described by

$$q_0 w_M w \vdash^*_{H'} \infty,$$

if M applied to w halts, and

$$q_0 w_M w \vdash^*_{H'} y_1 q_n y_2,$$

if M applied to w does not halt.

From H' we construct another Turing machine $\hat{H}$. This new machine takes as input w_M, copies it, and then behaves exactly like H'. Then the action of $\hat{H}$ is such that

$$q_0 w_M \vdash^*_{\hat{H}} q_0 w_M w_M \vdash^*_{\hat{H}} \infty,$$

if M applied to w_M halts, and

$$q_0 w_M \vdash^*_{\hat{H}} q_0 w_M w_M \vdash^*_{\hat{H}} y_1 q_n y_2,$$

if M applied to w_M does not 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 $\hat{w}$. From the above, identifying M with $\hat{H}$, we get

$$q_0\hat{w} \vdash^*_{\hat{H}} \infty,$$

if $\hat{H}$ applied to $\hat{w}$ halts, and

$$q_0\hat{w} \vdash^*_{\hat{H}} y_1 q_n y_2,$$

if $\hat{H}$ applied to $\hat{w}$ does not halt. This is clearly nonsense. The contradiction tells us that our assumption of the existence of H, and hence the assumption of the decidability of the halting problem, must be false. ■