Halting problem undecidability and infinitely nested simulation (V4)

A Simulating Halt Decider (SHD) computes the mapping from its input to its own accept or reject state based on whether or not the input simulated by a UTM would reach its final state in a finite number of simulated steps.

A halt decider (because it is a decider) must report on the behavior specified by its finite string input. This is its actual behavior when it is simulated by the UTM contained within its simulating halt decider while this SHD remains in UTM mode.

It is self-evidently correct that when-so-ever a simulating halt decider must abort the simulation of its input to prevent its infinite simulation that this input specifies an infinite sequence of configurations.

Simulating halt decider H performs a pure simulation of its input as if it was a UTM unless and until it detects an infinitely repeating pattern. Then it aborts the simulation of its input and transitions to its final reject state. Otherwise H transitions to its accept state when its simulation ends.

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. **A copy of Linz H is embedded at $Ĥ.q_x$**

$Ĥ.q_0$ ⟨Ĥ⟩ ⊢* $Ĥ.q_x$ ⟨Ĥ⟩ ⟨Ĥ⟩ ⊢* $Ĥ.q_y$ ∞
If the pure simulation of ⟨Ĥ⟩ ⟨Ĥ⟩ by embedded_H would reach its final state.

$Ĥ.q_0$ ⟨Ĥ⟩ ⊢* $Ĥ.q_x$ ⟨Ĥ⟩ ⟨Ĥ⟩ ⊢* $Ĥ.q_n$
If the pure simulation of ⟨Ĥ⟩ ⟨Ĥ⟩ by embedded_H would never reach its final state.

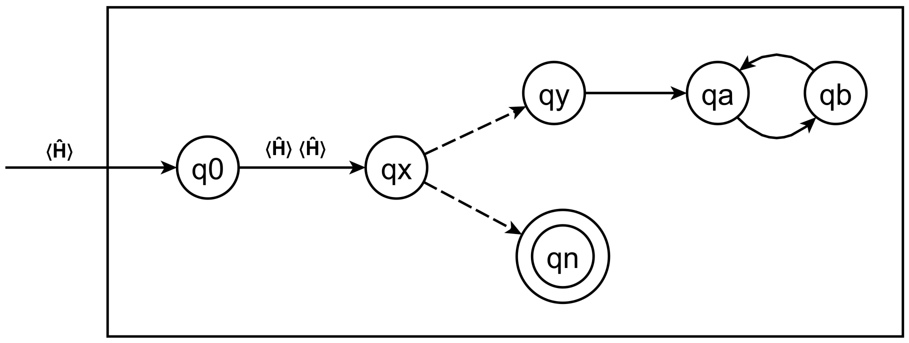


Figure 12.3 Turing Machine Ĥ applied to ⟨Ĥ⟩

When Ĥ is applied to **⟨Ĥ⟩**
Ĥ copies its input $⟨Ĥ_0⟩$ to $⟨Ĥ_1⟩$ then embedded_H simulates $⟨Ĥ_0⟩$ $⟨Ĥ_1⟩$

Then these steps would keep repeating:
$Ĥ_0$ copies its input $⟨Ĥ_1⟩$ to $⟨Ĥ_2⟩$ then embedded_H_0 simulates $⟨Ĥ_1⟩$ $⟨Ĥ_2⟩$
$Ĥ_1$ copies its input $⟨Ĥ_2⟩$ to $⟨Ĥ_3⟩$ then embedded_H_1 simulates $⟨Ĥ_2⟩$ $⟨Ĥ_3⟩$
$Ĥ_2$ copies its input $⟨Ĥ_3⟩$ to $⟨Ĥ_4⟩$ then embedded_H_2 simulates $⟨Ĥ_3⟩$ $⟨Ĥ_4⟩$...

The above shows that the simulated input to embedded_H never reaches its own final state whether or not its simulation is aborted.
(a) If the simulation is not aborted the above sequence never ends.
(b) If the simulation is aborted the entire chain of recursive simulations immediately stops.

In no case does the simulated input ⟨Ĥ⟩ ⟨Ĥ⟩ ever reach its final state ⟨Ĥ⟩.qn thus never meets the Linz definition of halting:

computation that halts … the Turing machine will halt whenever it enters a final state. (Linz:1990:234) Thus if embedded_H rejects its input it is necessarily correct.

Because all halt deciders are deciders they compute the mapping from their input finite strings to their own accept or reject state. Halt deciders (because they are deciders) do not compute any mappings from non-finite string non-inputs.

No halt decider ever determines the halt status of the computation that contains its actual self thus embedded_H does not compute the mapping from Ĥ ⟨Ĥ⟩ because it is neither an input nor a finite string.

Even Linz was confused by this. embedded_H is not supposed to report on itself or the computation that it is contained within.

In the conclusion of his proof Linz said that when the copy of Linz H embedded at Ĥ.q_x transitions to Ĥ.q_n on the basis that its input: ⟨Ĥ⟩ ⟨Ĥ⟩ never halts that this forms a contradiction with the fact that Ĥ ⟨Ĥ⟩ halts. This assumes that a halt decider must compute the halt status of the computation that contains itself.

As long as it is verified that the simulated input to embedded_H cannot reach its final state then we know that this simulated input cannot meet the Linz definition of halting.

As long as we know that this simulated input cannot meet the Linz definition of halting we know that this input specifies a non-halting sequence of configurations.

As long as we know that this input specifies a non-halting sequence of configurations then we know that embedded_H would be correct to reject this input.

By these two principles this proof is validated:
(1) A halt decider (because it is a decider) must report on the behavior specified by its finite string input. A decider computes the mapping from its input finite strings to an accept or reject state.

(2) The behavior specified by this input is the actual behavior of this input when simulated by the UTM contained within its simulating halt decider. Denying this point denies that the execution of a Turing machine is computationally equivanet to the simulation of the Turing machine description of this same machine.

Appendix: Peter Linz Halting Problem Proof

Definition 12.1

Let w_M describe a Turing machine $M = (Q, \Sigma, \Gamma, \delta, q_0, \square, F)$, and let w be any element of Σ^+. A solution of the halting problem is a Turing machine H, which for any w_M and w, performs the computation

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

if M applied to w halts, and

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

if M applied to w does not halt. Here q_y and q_n are both final states of H.

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

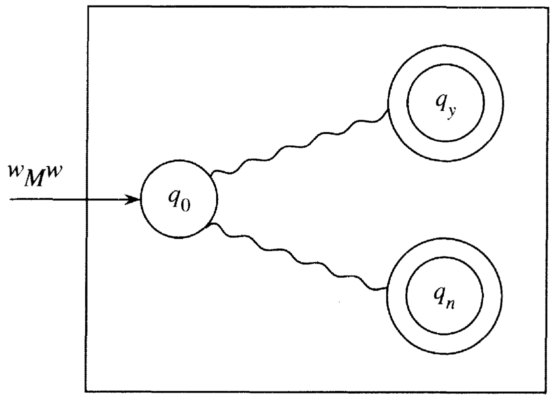
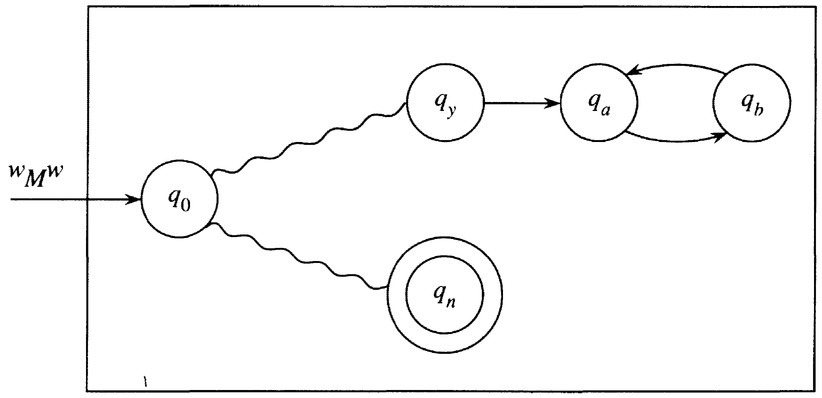


Figure 12.2



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

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

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.

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. ■

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