the Halting Problem

Will a given TM halt on a given input?

ie. Given as input:

- code(S) for a standard TM S
- a word w of C

..can we determine whether S halts and succeeds on input w?

We assume that we have a TM which determines whether S halts and succeeds - and derive a contradiction

Assumption is wrong

There is no such TM
The Halting Problem is unsolvable
**Formal specification of the Halting Problem:**

Define $h: \mathbb{C}^* \Rightarrow \mathbb{C}^*$ such that

$$h(x) =
\begin{cases} 
1 & \text{if } x = \text{code}(S)*w \text{ for a standard TM, } S, \text{ and } S \text{ halts and succeeds on input } w. \\
0 & \text{if } x = \text{code}(S)*w \text{ for some } S, w, \text{ and } S \text{ does not Halt and Succeed on input } w. \\
\text{undefined} & \text{if } x \text{ is not of the form code}(S)*w \text{ for any standard TM, } S \text{ and input } w.
\end{cases}$$

$h$ is a partial function $\mathbb{C}^* \Rightarrow \mathbb{C}^*$

**is there a TM $H$ such that $f_H = h$?**

Such a TM would solve the Halting Problem.
Proof of the Halting Problem

assume Turing Machine H s.t. \( f_H = h \)

define a partial function \( g : C^* \rightarrow C^* \) such that

\[
g(w) = 1 \text{ if } h(w*w) = 0 \\
\text{undefined otherwise}
\]

Let \( M \) be a TM with \( f_M = g \)

\( M \) has a code, \( \text{code}(M) \)

[we know how to encode the alphabet if \( M \) is not standard, using only characters of \( C \).]

Consider \( g(\text{code}(M)) \). It either has value 1 or is undefined.
1. Suppose $g(\text{code}(M)) = 1$
   $\Rightarrow h(\text{code}(M)*\text{code}(M)) = 0$ by defn. of $g$
   $\Rightarrow M$ does not Halt and Succeed on input $\text{code}(M)$ by defn. of $h$
   $\Rightarrow f_M(\text{code}(M))$ is undefined by defn. of Turing Machines
   $\Rightarrow g(\text{code}(M))$ is undefined…CONTRADICTION

2. Suppose $g(\text{code}(M))$ is not defined
   $\Rightarrow f_M(\text{code}(M))$ is not defined by defn of $M$
   $\Rightarrow M$ does not Halt and Succeed on input $\text{code}(M)$ by TM defn.
   $\Rightarrow h(\text{code}(M)*\text{code}(M)) = 0$ by defn of $h$
   $\Rightarrow g(\text{code}(M)) = 1$ …another CONTRADICTION

erroneous assumption: $\exists H$

there is no $H \Rightarrow$ by the Church-Turing Thesis
the Halting problem is unsolvable
the Halting Problem diagrammatically.
we deduce that M has a code and input $\text{code}(M)$ to M

```
M

assume input is $w \in C$

Add *w after w on the tape

(input w*w to H)

Halt & Succeed

\text{0 on tape} \quad 1 \text{ on tape} \quad \text{Halt & Fail}

H
```

**M Halt & Succeeds**

iff output of H is 0

iff $h(\text{code}(M) \ast \text{code}(M)) = 0$

iff M does not H & S on input $\text{code}(M)$

**M Halt & Fails**

iff output of H is 1

iff $h(\text{code}(M) \ast \text{code}(M)) = 1$

iff M H & S on input $\text{code}(M)$.

CONTRADICTIONS
Consequences of Halting Problem unsolvability:

• we cannot write a program “to see whether our programs loop”

  because this program (algorithm) would be implementable by a Turing Machine (by the Church-Turing thesis) …and we have just shown that no such TM exists.

• we can use the Halting Problem result to prove other unsolvability results..

if we can show that a solution to a new problem could be used to build a solution to the Halting Problem…we know this is impossible…so we conclude that the new problem must also be unsolvable.
Summary

We have proved ..

..by assuming that the Halting Problem had a Turing Machine (i.e. algorithmic) solution and demonstrating that this leads to a contradiction,

that no such TM exists and

therefore the Halting Problem is unsolvable..

there is no algorithmic solution
The run-time function of a Turing Machine

\[ M = (Q, \Sigma, \Gamma, q_0, \delta, F) \]

for input words \( w \) of length \( n \) (\( n = 1, 2, 3.. \)):

\( M \) runs a varying number of steps for various words \( w \) of length \( n \).

Define

\[ \text{time}_M (n) = \text{length of longest run of } M \text{ for input of length } n \]

the function

\[ \text{time}_M (n) : \{0, 1, 2, ..\} \Rightarrow \{0, 1, 2, ..., \infty\} \]

is the run-time function of \( M \).
We measure the **complexity** of a Turing machine by the order of its time function. Here we just investigate the running time in terms of the number of ‘steps’, or \(\delta\)-function entries executed during a run. This will be infinite if the TM does not halt. (see Part III, lectures 16-18).

Time functions may be:
- **linear**: \(\text{time}_M(n) = an+b\)
- **quadratic**: \(\text{time}_M(n) = an^2+bn+c\)
- **logarithmic**: \(\text{time}_M(n) = \log n + b\)
- **log linear**: \(\text{time}_M(n) = \log n + b\)
- **exponential**: \(\text{time}_M(n) = ba^n + c \) or \(ae^{n+b}\)

**Polynomial**: \(\text{time}_M(n) = a_1n^p + a_2n^{p-1} + a_3n^{p-2} + \ldots + a_p n + c\)

An important property of a TM is whether it runs in polynomial time. We describe Polynomial (or p-time) TMs as **fast**.
Time function for the **Tail** Turing machine:

time\text{Tail}(n) - longest run of Tail on input of length n.

Input length 0 - 1 step

1 - 2 steps - set squ 0 to ^, move right, read ^ & go into halting state.

3…set squ 0 to ^, move right: for each of n -1 symbols, move left, write symbol and move right;

1 step to move into halting state when symbol read = ^.

time\text{Tail}(n) = 1+3(n-1) + 1 for n>0. time\text{Tail}(0) = 1.

\begin{align*}
\delta\text{-function} \\
\delta(q_0,a) &= (q_1,^,1) \\
\delta(q_0,b) &= (q_1,^,1) \\
\delta(q_1,a) &= (q_a,^,-1) \\
\delta(q_1,b) &= (q_b,^,-1) \\
\delta(q_a,^) &= (q_2,a,1) \\
\delta(q_b,^) &= (q_2,b,1) \\
\delta(q_2,^) &= (q_1,^,1) \\
\delta(q_1,^) &= (q_3,^,0)
\end{align*}