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: C* $\Rightarrow$C* such that

h(x) = 1 if x =code(S)*w for a standard TM, S, and
S halts and succeeds on input w.
= 0 if x=code(S)*w for some S, w, and
S does not Halt and Succeed
on input w

is undefined if x is not of the form code(S)*w for any
standard TM, S and input w.

h is a partial function C*$\Rightarrow$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. $f_{\mathbf{H}} = h$

define a partial function g: g: $C^* \Rightarrow C^*$ such that

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

Let M be a TM with $f_{\mathbf{M}}=g$
M has a code, code(M)
[we know how to encode the alphabet if M is not standard, using only characters of C.]

Consider g(code(M))..it either has value 1 or is undefined..

1. Suppose g(code(M)) = 1
   ⇒h(code(M)*code(M)) = 0   by defn. of g
   ⇒ M does not Halt and Succeed on input code(M) by defn. of h
   ⇒ $f_{\mathbf{M}}$(code(M)) is undefined by defn. of Turing Machines
   ⇒ g(code(M)) is undefined…**CONTRADICTION**

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

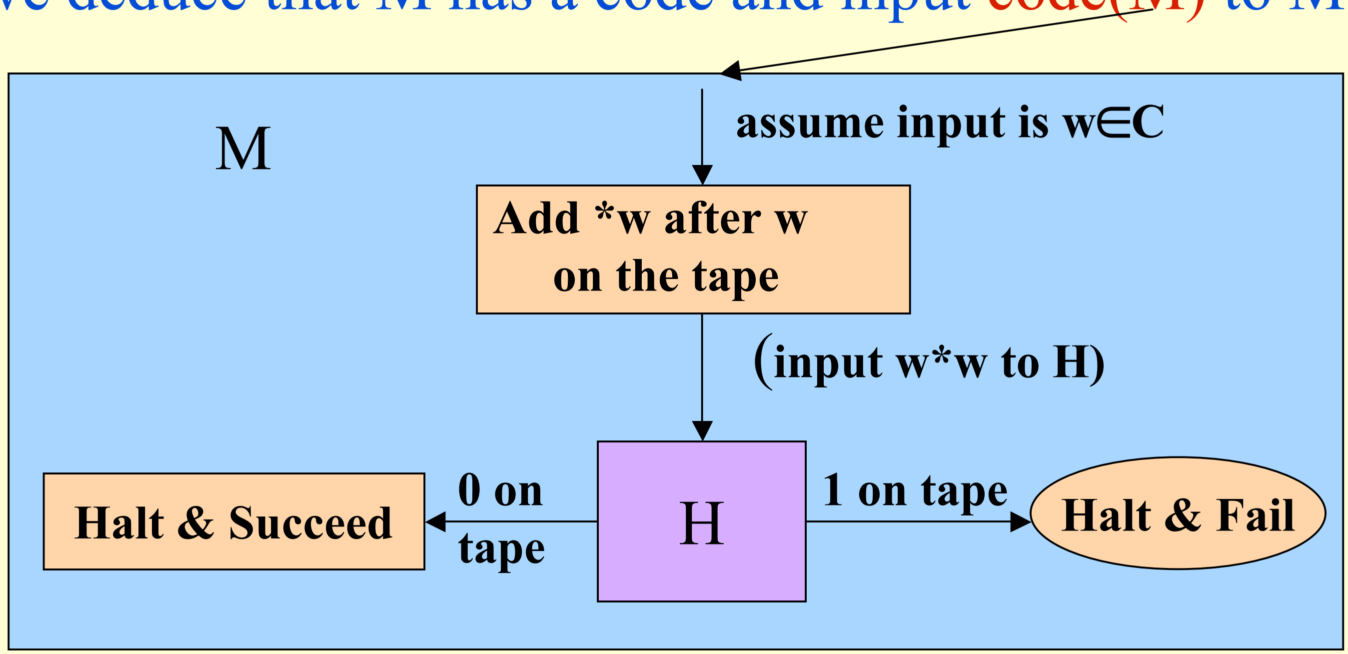
⇒ **erroneous assumption: ∃ H**

⇒ **there is no H ⇒** by the Church-Turing Thesis

the Halting problem is unsolvable

the Halting Problem diagrammatically..

we deduce that M has a code and input code(M) to M



M **Halts & Succeeds** iff output of H is 0
iff h(code(M)*code(M))=0
iff M **does not H & S** on input code(M)

M **Halts & Fails** iff output of H is 1
iff h(code(M)*code(M))=1
iff M **H & S** on input 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

$M = (Q, \sum, I, q_0, \delta, F)$

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

$time_M$ (n) = length of longest run of M for input of length n

the function

$time_M(n) : \{0, 1, 2, ..\} \Rightarrow \{0, 1, 2, \ldots, \infty\}$

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 δ-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: $\text{time}_M(n) = a\log n+b$

log linear: $\text{time}_M(n)\ \ an\log n+b$

exponential: $\text{time}_M(n) = ba^n+c$ or ae^{n+b}

Polynomial: $\mathbf{time_M(n) = a_1n^p+a_2n^{p-1}+a_3n^{p-2}+\ldots+a_pn+c}$

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:

$time_{Tail}(n)$ - longest run of Tail on input of length n.

Input length 0 - 1 step

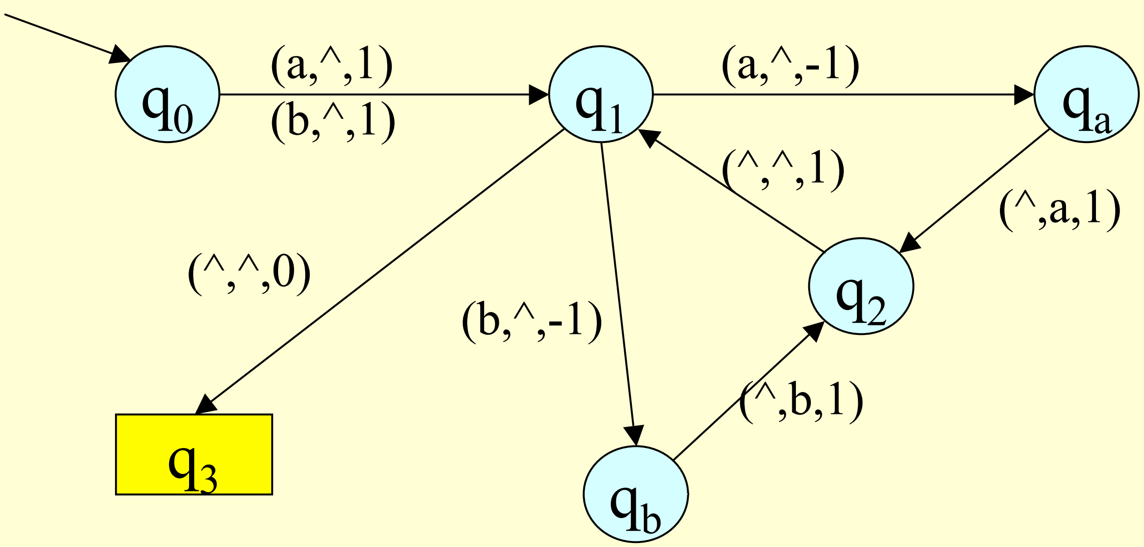
1 - 2 steps - set squ 0 to ^, move right, read ^ & go into halting state.

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

move right: 3 steps

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

$time_{Tail}(n) = 1+3(n-1) + 1$ for $n>0$. $time_{Tail}(0) = 1$.



δ-function

$\delta(q_0,a) = (q_1,\wedge,1)$

$\delta(q_0,b) = (q_1,\wedge,1)$

$\delta(q_1,a) = (q_a,\wedge,-1)$

$\delta(q_1,b) = (q_b,\wedge,-1)$

$\delta(q_a,\wedge) = (q_2,a,1)$

$\delta(q_b,\wedge) = (q_2,b,1)$

$\delta(q_2,\wedge) = (q_1,\wedge,1)$

$\delta(q_1,\wedge) = (q_3,\wedge,0)$

continued..