The halting problem

INSTANCE: A binary Turing machine M, and an input $x \in \{0,1\}^*$.

QUESTION: Does M halt on input x?

What we will do:

1. Make this question precise by phrasing it as a language recognition problem.
2. Show that the language is not recognized by *any* TM.
3. For amusement, produce a function with a mind blowing growth rate.

Recall: For every Turing machine M with input alphabet $\{\mathtt{0}, \mathtt{1}\}$, there is a binary Turing machine $\widehat{M}$ that is equivalent to M: on every input, $\widehat{M}$ halts if and only if M halts, and $\widehat{M}$ accepts if and only if M accepts. (Note: *tape* alphabet of M is unrestricted.)

Example: TM M with

$$\Sigma = \{\,\mathtt{0}, \mathtt{1}\,\}$$
$$\Gamma = \{\,\mathtt{0}, \mathtt{1}, \$, \#\,\}.$$

Encode:

$$\begin{array}{ccccc} \mathtt{0} & \mathtt{1} & \$ & \# & \not{b} \\ \downarrow & \downarrow & \downarrow & \downarrow & \downarrow \\ \mathtt{00} & \mathtt{01} & \mathtt{10} & \mathtt{11} & \not{b}\not{b} \end{array}$$

Simulate M by machine $\widehat{M}$ with input alphabet $\{\,\mathtt{0}, \mathtt{1}\,\}$ and tape alphabet $\{\,\mathtt{0}, \mathtt{1}, \not{b}\,\}$.

M, $\widehat{M}$ get same input:

0	1	1	0	$\not{b}$	$\cdots$

$\widehat{M}$ first encodes input and then simulates M:

(i) Mark first square for later:

$\not{b}$	0	1	1	0	$\not{b}$	$\cdots$

(ii) Encode each symbol by using repeated right shifts:

$\not{b}$	0	0	0	1	0	1	0	0	$\not{b}$	$\cdots$

(iii) Shift left at the end:

0	0	0	1	0	1	0	0	$\not{b}$	$\cdots$
$\wedge$									

Now simulate M by doing everything in blocks of 2:

- 2 steps to read,
- 2 steps to write.

Recall: Encodings of binary TMs presented to UTM in format

`0001011*0010111*01B1010*1000111`

Use of `*` and `B` is to help *us*, machines couldn't care less.

Encode as a binary string by mapping:

$$\begin{array}{cccc} \mathtt{0} & \mathtt{1} & \mathtt{B} & \mathtt{*} \\ \downarrow & \downarrow & \downarrow & \downarrow \\ \mathtt{00} & \mathtt{01} & \mathtt{10} & \mathtt{11} \end{array}$$

Now

`0001011*0010111*01B1010*1000111` $\longrightarrow$

`00000001000101110000010001010111`

`11000110010001001101000000010101`

What's the point? Encodings of TMs now look the same as inputs to TMs ('programs as data').

Note: We insist that final state is given binary code for 1.

Halting problem as language recognition:

For binary TM M use $\langle M \rangle$ to denote its encoding as a binary string.

Define $L_{\mathsf{halt}} \subset \{\, 0, 1, \$\, \}$ by:

$$L_{\mathsf{halt}} = \{\langle M \rangle \$ x \mid x \in \{0, 1\}^* \text{ and } M \text{ halts on input } x\}.$$

Note: Lots of words fail to be in L_{halt} because they are badly formatted; obviously we can recognize these (by a TM).

LEMMA The language L_{halt} is recursively enumerable.

Gives a *semi-decision* procedure.

Most important result result of this module:

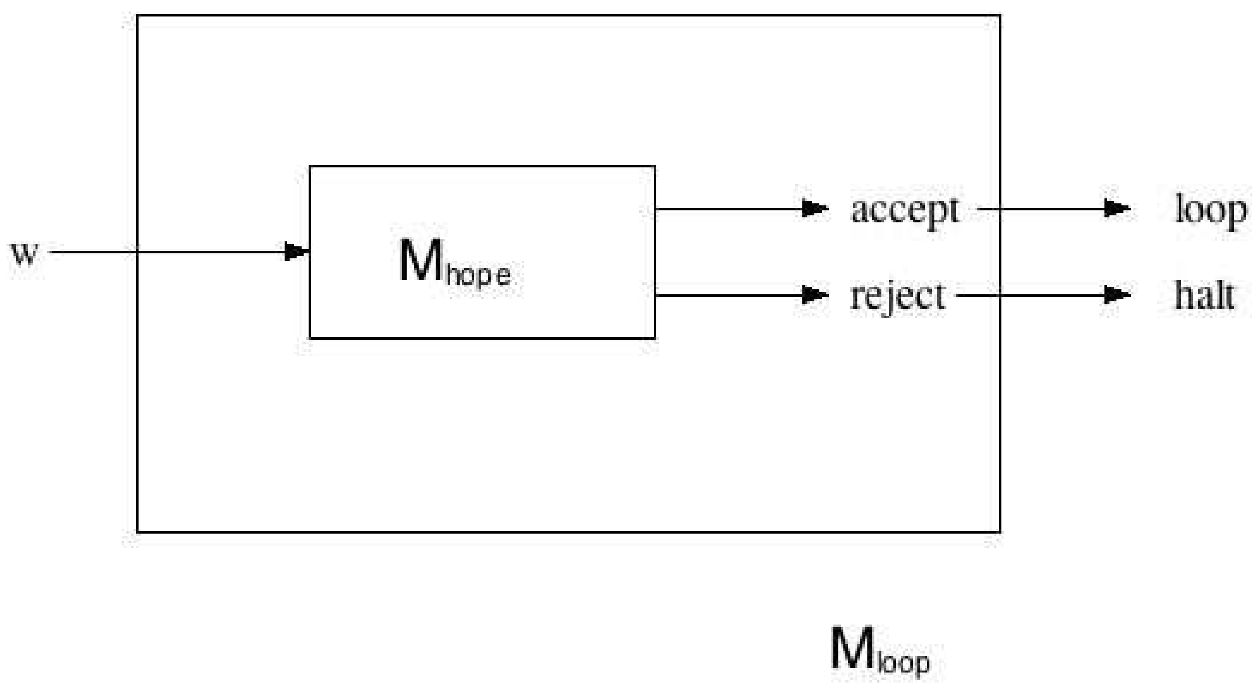
THEOREM The language L_{halt} is not recursive.

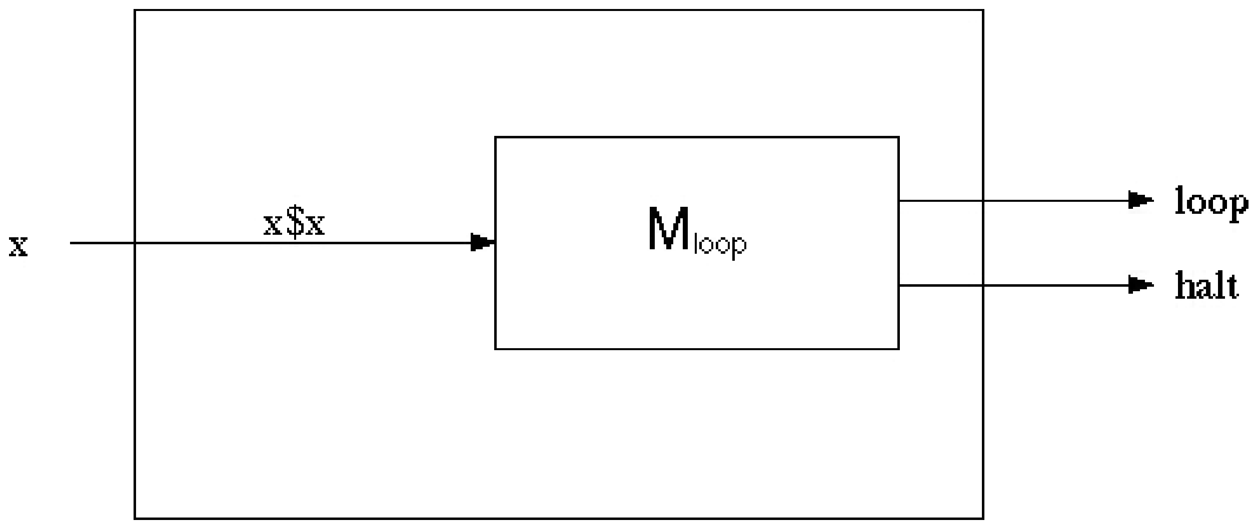
Dashes all hope of finding a *decision* procedure for the halting problem.

PROOF Suppose L_{halt} *is* recursive; so there is a TM M_{hope} that:

1. halts on *all* inputs,
2. accepts its input if and only if it is of form $\langle M \rangle \$ x$ and machine M halts on input x.

We will derive a contradiction.


w
M_{hope}
accept
reject
loop
halt
M_{loop}



Mdiag

M_{diag} has input alphabet $\{0, 1\}$. Transform it to equivalent binary TM M_{liar} with binary encoding $\langle M_{\text{liar}} \rangle$.

Now run M_{liar} on its own description:

$$
\begin{aligned}
& M_{\text{liar}} \text{ halts on input } \langle M_{\text{liar}} \rangle \\
& \qquad \iff M_{\text{hope}} \text{ rejects } \langle M_{\text{liar}} \rangle \$ \langle M_{\text{liar}} \rangle \\
& \qquad \iff M_{\text{liar}} \text{ does not halt on input } \langle M_{\text{liar}} \rangle.
\end{aligned}
$$

A contradiction!

Conclusion: M_{hope} does not exist, i.e., L_{halt} is not recursive.

Note: Constructive nature of proof.

An explosive function: M a binary TM, $x \in \{0,1\}^*$ an input to M.

$$T(M,x) = \begin{cases} \text{no. of transitions} & \text{if } M \text{ halts on } x, \\ \text{undefined} & \text{otherwise.} \end{cases}$$

Define $f : \mathbb{N} \to \mathbb{N}$ by:

$$f(n) = \begin{cases} 0 & \text{if } n = 0, \\ \max\{T(M,x) \mid & \\ \quad M \text{ halts on input } x & \\ \quad \text{and } \langle M \rangle \$ x \text{ has length } n\} & \text{if } n > 0. \end{cases}$$

Note: f is a perfectly well defined total function:

- given $n > 0$ have only finitely many $\langle M \rangle \$ x$ of length n and at least one always halts (the empty machine).

Question: Is there a TM transducer that computes f?

Suppose there is, call it M_f. Then can solve halting problem as follows (using a 2-tape TM).

Given M and input x:

1. Work out length of input, say n, and write it on second tape.

2. Use M_f to compute $f(n)$ on the second tape.

3. Simulate M on x for at most $f(n)$ transitions. If M halts then halt and accept else halt and reject (if M doesn't halt within $f(n)$ steps it's not going to halt anyway).

Suppose $g : \mathbb{N} \to \mathbb{N}$ is any other function such that

$$g(n) \geq f(n), \quad \text{for all } n \in \mathbb{N}.$$

Similar argument shows g is not computable.

Conclusion: f grows faster than *every* computable function!