A little summary:

L is recursively enumerable (r.e.)

$\Longleftrightarrow$ $L = L(M)$ for a deterministic TM M

$\Longleftrightarrow$ $L = L(M)$ for a non-deterministic TM M

$\Longleftrightarrow$ $\exists$ a deterministic semi-algorithm for the decision problem
"Is a given word w in L ?"

$\Longleftrightarrow$ $\exists$ a non-deterministic semi-algorithm for the decision problem
"Is a given word w in L ?"

A non-deterministic semi-algorithm for a decision problem **guesses a certificate** x for the positivity of the given instance w, and verifies that x indeed proves that w is positive. (If w is positive then such a certificate x must exist; if w is negative then such a certificate x may not exist.)

L is recursive (Rec)

$\iff$ $L = L(M)$ for a deterministic TM M that halts with every input

$\iff$ $\exists$ a deterministic algorithm for the decision problem
"Is a given word w in L ?"

$\iff$ $L = L(M_1)$ and $\Sigma^* \setminus L = L(M_2)$ for non-deterministic TMs M_1 and M_2

$\iff$ $\exists$ non-deterministic semi-algorithms for the decision problems
"Is a given word w in L ?" and "Is a given word w in $\Sigma^* \setminus L$?"

When a decision problem P is semi-decidable we say that the **positive instances** of P are semi-decidable. When the complement of a decision problem P is semi-decidable we say that the **negative instances** of P are semi-decidable.

So a decision problem is decidable iff its positive instances are semi-decidable and its negative instances are semi-decidable.

L is recursive (Rec)

$\Longleftrightarrow$ $L = L(M)$ for a deterministic TM M that halts with every input

$\Longleftrightarrow$ $\exists$ a deterministic algorithm for the decision problem
"Is a given word w in L ?"

$\Longleftrightarrow$ $L = L(M_1)$ and $\Sigma^* \setminus L = L(M_2)$ for non-deterministic TMs M_1 and M_2

$\Longleftrightarrow$ $\exists$ non-deterministic semi-algorithms for the decision problems
"Is a given word w in L ?" and "Is a given word w in $\Sigma^* \setminus L$?"

Thus we sometimes prove the decidability of P by proving that

- there exist certificates for the positivity of positive instances, and
- there exist certificates for the negativity of negative instances.

The validity of certificates must be effectively checkable.

Example. The decision problem: "Does a given directed graph contain a cycle?" is decidable.

Positive instances: A certificate is a sequence $v_1, v_2, \ldots, v_k$ of vertices that forms a cycle. (To verify the certificate check that for every i there is an edge $v_i \longrightarrow v_{i+1}$, and that there also is the edge $v_k \longrightarrow v_1$.)

Negative instances: A certificate is a permutation $v_1, v_2, \ldots, v_n$ of all vertices that is a topological sort. (To verify the certificate check that it is a permutation and that for all $i < j$ there is no edge $v_j \longrightarrow v_i$.)

Example. The decision problem: "Does a given directed graph contain a cycle?" is decidable.

Positive instances: A certificate is a sequence $v_1, v_2, \ldots, v_k$ of vertices that forms a cycle. (To verify the certificate check that for every i there is an edge $v_i \longrightarrow v_{i+1}$, and that there also is the edge $v_k \longrightarrow v_1$.)

Negative instances: A certificate is a permutation $v_1, v_2, \ldots, v_n$ of all vertices that is a topological sort. (To verify the certificate check that it is a permutation and that for all $i < j$ there is no edge $v_j \longrightarrow v_i$.)

(In practice one would rather use **depth-first-search** to check for the presence of cycles in a given directed graph. This algorithm is very fast: linear time in the size of the graph.)

Decision problems concerning Turing machines

In the following we consider decision problems whose instances involve Turing machines. We prove that many such problems are undecidable (even non-semi-decidable).

So there does not exist a Turing machine that recognizes the (encodings of) positive instances of these problems.

We start by fixing our encoding scheme, *i.e.*, the way we represent Turing machines as words.

Let us consider now only Turing machines with the input alphabet $\{a, b\}$.

By renaming the states and tape letters we may assume that the state set is

$$Q = \{q_1, q_2, \ldots, q_n\}$$

and that the tape alphabet is

$$\Gamma = \{X_1, X_2, \ldots, X_m\}$$

The input alphabet $\{a, b\}$ and the blank symbol B must be part of the tape alphabet, so we assume that $m \geq 3$ and

$$X_1 = a, X_2 = b, X_3 = B$$

We may also assume that q_1 is the initial state and q_n is the final state of the machine, and that $n \geq 2$.

So we restrict our instances to Turing machines $M = (Q, \Sigma, \Gamma, \delta, q_1, B, q_n)$ where

$$\begin{aligned} Q &= \{q_1, q_2, \ldots, q_n\}, \\ \Sigma &= \{a, b\}, \\ \Gamma &= \{X_1, X_2, \ldots, X_m\} \text{ with} \\ & \quad X_1 = a, \\ & \quad X_2 = b, \\ & \quad X_3 = B, \end{aligned}$$

and $n \geq 2$ and $m \geq 3$.

It is clear that such Turing machines recognize all recursively enumerable languages over the alphabet $\{a, b\}$, and that halting machines of this kind recognize all recursive languages over $\{a, b\}$.

Encoding of M as a word $\langle M \rangle \in \{a, b\}^*$

Let us number L and R as directions 1 and 2:

$$\begin{aligned} D_1 &= L, \\ D_2 &= R. \end{aligned}$$

An arbitrary transition

$$\delta(q_i, X_j) = (q_k, X_l, D_s)$$

by the machine M is encoded as the word

$$a^i \; b \; a^j \; b \; a^k \; b \; a^l \; b \; a^s$$

Then we encode the machine M as the word

$$\langle M \rangle = bbb \; a^n \; bb \; a^m \; bb \; \text{code}_1 \; bb \; \text{code}_2 \; bb \ldots bb \; \text{code}_r \; bbb$$

where $\text{code}_1, \text{code}_2, \ldots, \text{code}_r$ are encodings of all the defined transitions $\delta(q_i, X_j)$, listed in the **lexicographic order** of i, j.

Example. Let

$$M = (\{q_1, q_2, q_3\}, \{a, b\}, \{a, b, B, X_4\}, \delta, q_1, B, q_3)$$

with the transitions

$$\begin{aligned}
\delta(q_1, a) &= (q_1, b, R), \\
\delta(q_1, B) &= (q_2, B, L), \\
\delta(q_2, b) &= (q_1, X_4, R), \\
\delta(q_2, B) &= (q_3, a, R), \\
\delta(q_2, X_4) &= (q_1, B, L).
\end{aligned}$$

Then

$\langle M \rangle =$

Example. Let

$$M = (\{q_1, q_2, q_3\}, \{a, b\}, \{a, b, B, X_4\}, \delta, q_1, B, q_3)$$

with the transitions

$$\begin{aligned}
\delta(q_1, a) &= (q_1, b, R), \\
\delta(q_1, B) &= (q_2, B, L), \\
\delta(q_2, b) &= (q_1, X_4, R), \\
\delta(q_2, B) &= (q_3, a, R), \\
\delta(q_2, X_4) &= (q_1, B, L).
\end{aligned}$$

Then

$$\langle M \rangle = \textit{bbbaaabbaaaabb abababaabaa bb abaaabaabaaaba bb aabaababaaaabaa bb aabaaabaaababaa bb aabaaaababaaaba bbb}$$

The set of valid encodings of Turing machines is recursive: there exists an algorithm that checks whether a given word w is an encoding of some Turing machine.

Lemma. The language

$$L_{enc} = \{\ w \in \{a,b\}^* \mid\ w = \langle M \rangle \text{ for some Turing machine } M\ \ \}$$

is recursive.

Proof.

Proof that L_{enc} is recursive:

There is a TM that verifies the following facts about the given input word w:

- Word w has the correct form

$$bbb\ a^n\ bb\ a^m\ bb\ \text{code}_1\ bb\ \text{code}_2\ bb \ldots bb\ \text{code}_r\ bbb$$

  for some $n \geq 2$ and $m \geq 3$, where each code_t is a word from $a^+ba^+ba^+ba^+ba^+$. (Such words form even a regular language.)

Proof that L_{enc} is recursive:

There is a TM that verifies the following facts about the given input word w:

- Word w has the correct form

$$bbb\ a^n\ bb\ a^m\ bb\ \text{code}_1\ bb\ \text{code}_2\ bb \ldots bb\ \text{code}_r\ bbb$$

  for some $n \geq 2$ and $m \geq 3$, where each code_t is a word from $a^+ba^+ba^+ba^+ba^+$. (Such words form even a regular language.)

- Each $\mathbf{code}_t = a^i\ b\ a^j\ b\ a^k\ b\ a^l\ b\ a^s$ satisfies the conditions

$$\begin{array}{rcl} 1 \leq i & \leq & n-1, \\ 1 \leq k & \leq & n, \\ 1 \leq j, l & \leq & m, \\ 1 \leq s & \leq & 2. \end{array}$$

  Then each code_t is a valid encoding of some transition $\delta(q_i, X_j) = (q_k, X_l, D_s)$.

Proof that L_{enc} is recursive:

There is a TM that verifies the following facts about the given input word w:

- Word w has the correct form

$$bbb\ a^n\ bb\ a^m\ bb\ \text{code}_1\ bb\ \text{code}_2\ bb \ldots bb\ \text{code}_r\ bbb$$

  for some $n \geq 2$ and $m \geq 3$, where each code_t is a word from $a^+ba^+ba^+ba^+ba^+$. (Such words form even a regular language.)

- Each $\mathbf{code}_t = a^i\ b\ a^j\ b\ a^k\ b\ a^l\ b\ a^s$ satisfies the conditions

$$\begin{array}{rcl} 1 \leq i & \leq & n-1, \\ 1 \leq k & \leq & n, \\ 1 \leq j, l & \leq & m, \\ 1 \leq s & \leq & 2. \end{array}$$

  Then each code_t is a valid encoding of some transition $\delta(q_i, X_j) = (q_k, X_l, D_s)$.

- For all consecutive

$$\text{code}_t = a^i\ b\ a^j\ b\ a^k\ b\ a^l\ b\ a^s \text{ and } \text{code}_{t+1} = a^{i'}\ b\ a^{j'}\ b\ a^{k'}\ b\ a^{l'}\ b\ a^{s'}$$

  we have either $i < i'$, or both $i = i'$ and $j < j'$. This guarantees the proper lexicographic ordering of the transitions. This also guarantees that the machine is deterministic.

Proof that L_{enc} is recursive:

There is a TM that verifies the following facts about the given input word w:

- Word w has the correct form

$$bbb\ a^n\ bb\ a^m\ bb\ \text{code}_1\ bb\ \text{code}_2\ bb \ldots bb\ \text{code}_r\ bbb$$

  for some $n \geq 2$ and $m \geq 3$, where each code_t is a word from $a^+ba^+ba^+ba^+ba^+$. (Such words form even a regular language.)

- Each $\mathbf{code}_t = a^i\ b\ a^j\ b\ a^k\ b\ a^l\ b\ a^s$ satisfies the conditions

$$\begin{array}{rcl} 1 \leq i & \leq & n-1, \\ 1 \leq k & \leq & n, \\ 1 \leq j, l & \leq & m, \\ 1 \leq s & \leq & 2. \end{array}$$

  Then each code_t is a valid encoding of some transition $\delta(q_i, X_j) = (q_k, X_l, D_s)$.

- For all consecutive

$$\text{code}_t = a^i\ b\ a^j\ b\ a^k\ b\ a^l\ b\ a^s \text{ and } \text{code}_{t+1} = a^{i'}\ b\ a^{j'}\ b\ a^{k'}\ b\ a^{l'}\ b\ a^{s'}$$

  we have either $i < i'$, or both $i = i'$ and $j < j'$. This guarantees the proper lexicographic ordering of the transitions. This also guarantees that the machine is deterministic.

Input word w represents a Turing machine iff all three conditions are satisfied.

We define next the **complemented diagonal language**

$$L_d = \{\langle M \rangle \mid \ M \text{ does not accept the word } \langle M \rangle \ \}.$$

(Encodings of those Turing machines that do not accept their own encoding.)

The language L_d is a well defined language over the alphabet $\{a, b\}$. It is our first example of a language that is not recursively enumerable.

Theorem (Alan Turing 1936). The complemented diagonal language

$$L_d = \{\langle M \rangle \mid \ M \text{ does not accept the word } \langle M \rangle \ \}$$

is not recursively enumerable.

Proof.

Theorem (Alan Turing 1936). The complemented diagonal language

$$L_d = \{\langle M \rangle \mid \ M \text{ does not accept the word } \langle M \rangle \ \}$$

is not recursively enumerable.

Proof. An indirect proof: Suppose that, contrary to the claim, L_d is recognized by some Turing machine M_d.

Does M_d accept the input $\langle M_d \rangle$?

Theorem (Alan Turing 1936). The complemented diagonal language

$$L_d = \{\langle M \rangle \mid \ M \text{ does not accept the word } \langle M \rangle \ \}$$

is not recursively enumerable.

Proof. An indirect proof: Suppose that, contrary to the claim, L_d is recognized by some Turing machine M_d.

Does M_d accept the input $\langle M_d \rangle$?

- If M_d does not accept $\langle M_d \rangle$ then, by the definition of the language L_d, the word $\langle M_d \rangle$ is in L_d. But M_d recognizes L_d, so M_d accepts $\langle M_d \rangle$, a contradiction.

Theorem (Alan Turing 1936). The complemented diagonal language

$$L_d = \{\langle M \rangle \mid \ M \text{ does not accept the word } \langle M \rangle \ \}$$

is not recursively enumerable.

Proof. An indirect proof: Suppose that, contrary to the claim, L_d is recognized by some Turing machine M_d.

Does M_d accept the input $\langle M_d \rangle$?

- If M_d does not accept $\langle M_d \rangle$ then, by the definition of the language L_d, the word $\langle M_d \rangle$ is in L_d. But M_d recognizes L_d, so M_d accepts $\langle M_d \rangle$, a contradiction.
- If M_d accepts $\langle M_d \rangle$ then, by the definition of the language L_d, word $\langle M_d \rangle$ is not in L_d. But M_d recognizes L_d, so M_d does not accept $\langle M_d \rangle$, a contradiction.

Theorem (Alan Turing 1936). The complemented diagonal language

$$L_d = \{\langle M \rangle \mid \ M \text{ does not accept the word } \langle M \rangle \ \}$$

is not recursively enumerable.

Proof. An indirect proof: Suppose that, contrary to the claim, L_d is recognized by some Turing machine M_d.

Does M_d accept the input $\langle M_d \rangle$?

- If M_d does not accept $\langle M_d \rangle$ then, by the definition of the language L_d, the word $\langle M_d \rangle$ is in L_d. But M_d recognizes L_d, so M_d accepts $\langle M_d \rangle$, a contradiction.
- If M_d accepts $\langle M_d \rangle$ then, by the definition of the language L_d, word $\langle M_d \rangle$ is not in L_d. But M_d recognizes L_d, so M_d does not accept $\langle M_d \rangle$, a contradiction.

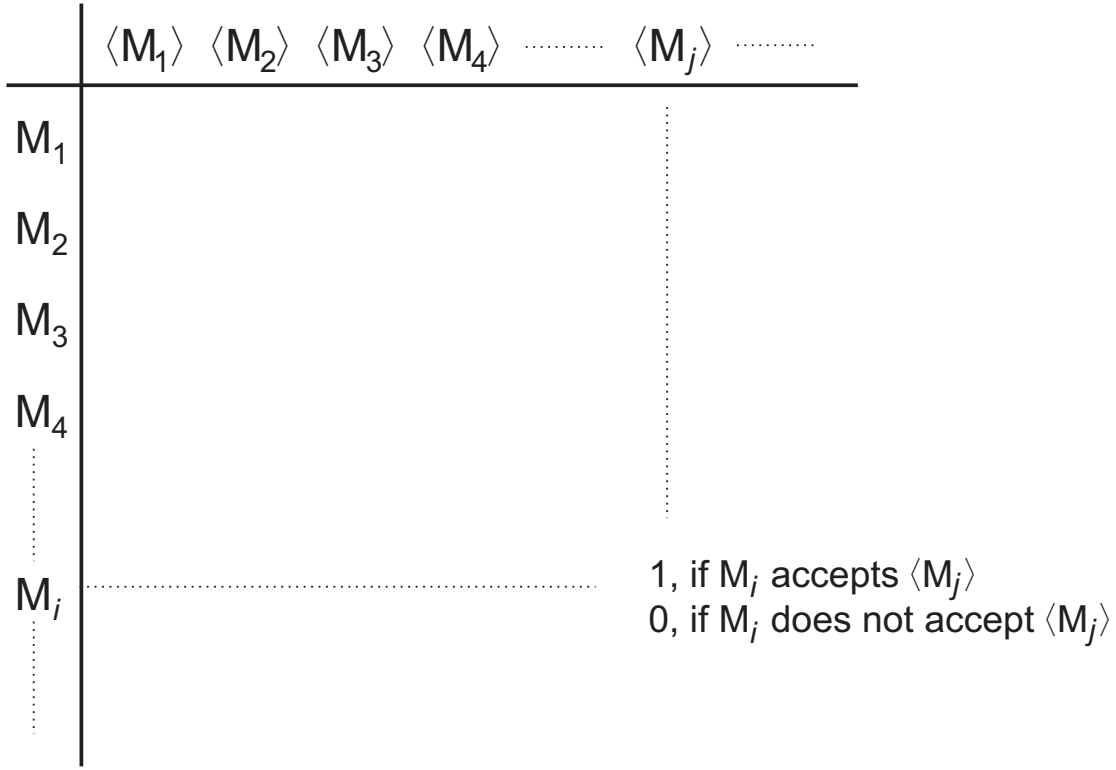
In both cases we reach a contradiction. Therefore M_d cannot exist.

The proof is, in fact, the powerful **diagonal** (or self-reference) argument used in many contexts in mathematics:

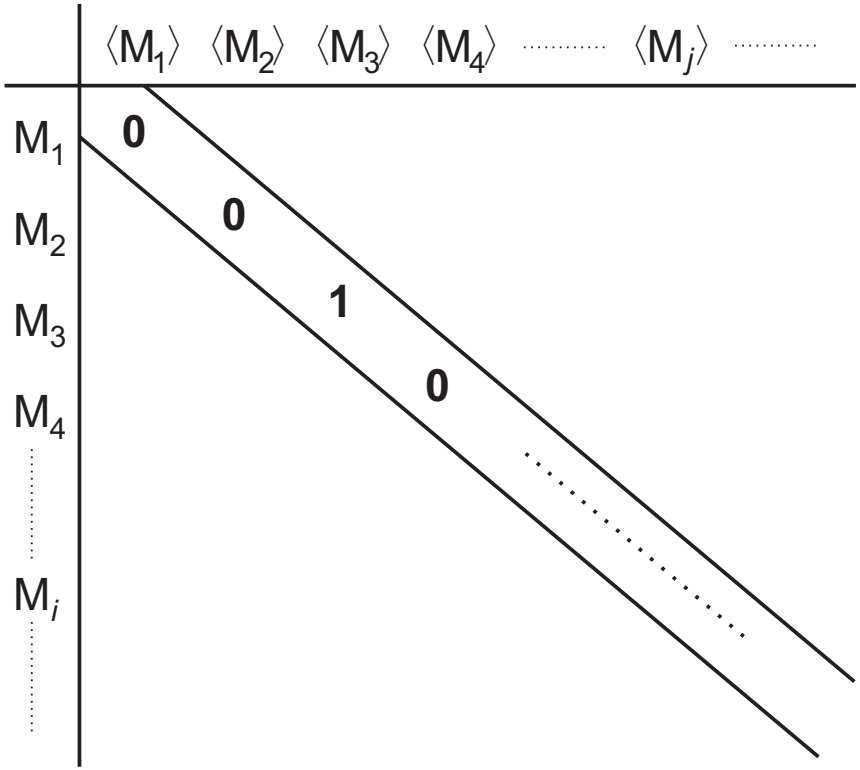
- G. Cantor (1891): the set of reals is not countable,
- K. Gödel (1931): first incompleteness theorem,
- Epimenides the Cretan: "*all Cretans are liars*"
- ...

The diagonal aspect of Turing's proof can be visualized as follows: Consider an arbitrary enumeration $M_1, M_2, \ldots$ of Turing machines with input alphabet $\{a, b\}$, and an an infinite 0/1-matrix whose rows are indexed by Turing machines M_i and whose columns are indexed by their encodings $\langle M_i \rangle$.

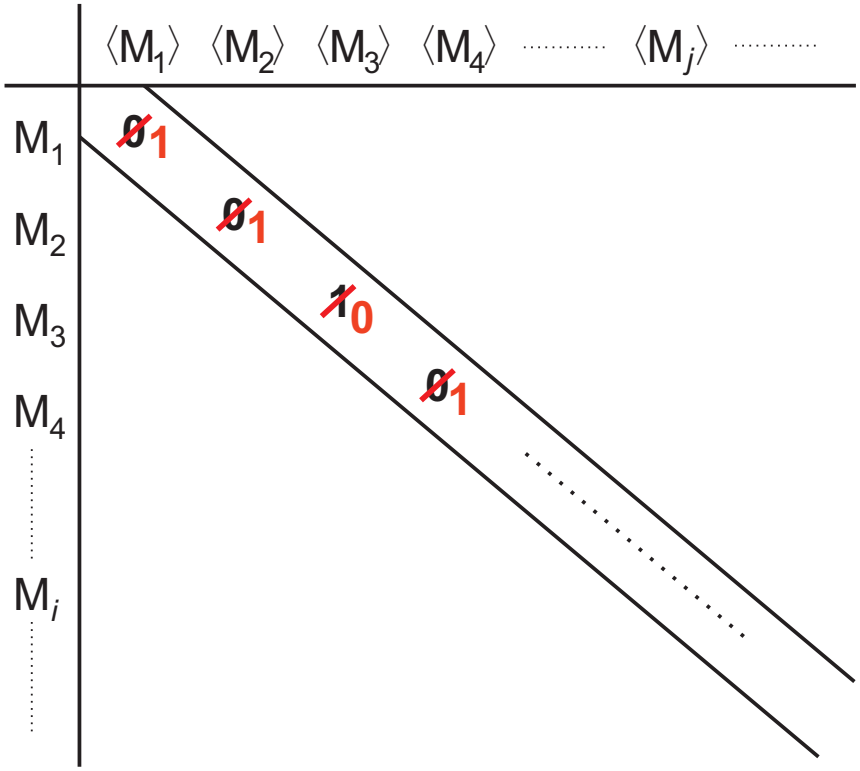
The entry $(M_i, \langle M_j \rangle)$ of the table is 1 iff M_i accepts word $\langle M_j \rangle$, so that the row indexed by M_i is the **characteristic sequence** identifying which words $\langle M_j \rangle$ are accepted by M_i:



Consider the sequence of bits along the diagonal $(M_i, \langle M_i \rangle)$, and complement this sequence by swapping each bit.



Consider the sequence of bits along the diagonal $(M_i, \langle M_i \rangle)$, and complement this sequence by swapping each bit.



The complemented diagonal cannot be identical to any row of the table. So a language whose characteristic sequence is the complemented diagonal is not recognized by any Turing machine.

Our L_d is this complemented diagonal language.

So the following decision problem is not semi-decidable (and hence undecidable):

"Given a TM M, is it true that $\langle M \rangle \notin L(M)$?"

This decision problem is rather artificial, but it serves as a **seed** from which we can conclude many other – more natural – problems to be undecidable. The method we use to obtain undecidability results is **reduction**.

As a first example of reductions, let us show that there is no algorithm to determine if a given Turing machine accepts a given input word:

Corollary. The language

$$L_u = \{\ \langle M \rangle \# w \mid \ M \text{ accepts the input word } w\ \}$$

is not recursive. More specifically, its complement is not recursively enumerable.

Proof.

As a first example of reductions, let us show that there is no algorithm to determine if a given Turing machine accepts a given input word:

Corollary. The language

$$L_u = \{\ \langle M\rangle \# w \mid\ \ M \text{ accepts the input word } w\ \}$$

is not recursive. More specifically, its complement is not recursively enumerable.

Proof. Suppose the contrary: There is a Turing machine $M_{\bar{u}}$ that recognizes the complement of L_u.

Then the following TM M_d recognizes L_d, contradicting Turing's theorem.

Machine M_d works as follows: On input w it

1. Checks whether w is a valid encoding of some Turing machine. This can be done effectively, as the language L_{enc} of valid encodings is recursive.

   If w is not a valid encoding, machine M_d halts in a non-final state.

2. If $w = \langle M\rangle$ is a valid encoding then M_d writes $w\#w = \langle M\rangle\#\langle M\rangle$ on the tape and starts $M_{\bar{u}}$ on this input. The final state is entered if and only if $M_{\bar{u}}$ accepts this word.

Such a Turing machine M_d exists if $M_{\bar{u}}$ exists. (It uses $M_{\bar{u}}$ as a subroutine.) Clearly M_d recognizes L_d. This is not possible so $M_{\bar{u}}$ cannot exist.

The reduction described a TM M_d that recognizes a known non-r.e. language L_d, using as a subroutine a hypothetical TM $M_{\bar{u}}$ that recognizes the complement of L_u.

(I did not provide a detailed transition function of M_d, but a precise enough construction of it so that it is clear that it exists if $M_{\bar{u}}$ exists.)

In the reduction I described a TM M_d that recognizes a known non-r.e. language L_d, using as a subroutine a hypothetical TM $M_{\bar{u}}$ that recognizes the complement of L_u.

(I did not provide a detailed transition function of M_d, but a precise enough construction of it so that it is clear that it exists if $M_{\bar{u}}$ exists.)

Here's a "high level" explanation of the reduction:

Suppose there is a semi-algorithm to test whether a given Turing machine does not accept a given word. This semi-algorithm then recognizes 0's in the infinite table discussed above. But it then also identifies entries 1 on the inverted diagonal, contradicting Turing's theorem.

Another (more complicated) example of a reduction.

Corollary. The language

$$L_{halt} = \{ \langle M \rangle \mid \ M \text{ halts when started on the blank tape } \}$$

is not recursive.

Proof.

Another (more complicated) example of a reduction.

Corollary. The language

$$L_{halt} = \{ \langle M \rangle \mid \ M \text{ halts when started on the blank tape } \}$$

is not recursive.

Proof. Let us do the reduction at the level of an informal algorithm, rather than explicitly constructing a Turing machine.

Suppose, on the contrary to the claim, that there exists an algorithm A_{halt} to determine if a given Turing machine halts when started on the blank tape.

Then the following algorithm $A_{\bar{u}}$ determines whether a given Turing machine M accepts a given word w, contradicting the previous corollary that states that such an algorithm does not exist. The algorithm $A_{\bar{u}}$ uses the hypothetical algorithm A_{halt} as a subroutine.

On input M and w the algorithm $A_{\bar{u}}$ **does the following**.

1. It **builds a new Turing machine** M' that works as follows:
   - First M' writes w on its tape. (This may be done using $|w|$ states.)
   - Then M' moves back to the first letter of w and enters the initial state of M.
   - From there on, transitions of machine M are used.
   - For each halting but non-final state of M the machine M' goes into a new looping state that makes the machine move to the right indefinitely without ever halting.

   Clearly, this M' **halts on the blank initial tape if and only if** M **accepts the input** w.

On input M and w the algorithm $A_{\bar{u}}$ **does the following**.

1. It **builds a new Turing machine** M' that works as follows:
   - First M' writes w on its tape. (This may be done using $|w|$ states.)
   - Then M' moves back to the first letter of w and enters the initial state of M.
   - From there on, transitions of machine M are used.
   - For each halting but non-final state of M the machine M' goes into a new looping state that makes the machine move to the right indefinitely without ever halting.

   Clearly, this M' **halts on the blank initial tape if and only if** M **accepts the input** w.

2. Algorithm $A_{\bar{u}}$ does not execute M'. Instead, it just **gives** M' **as an input to the hypothetical algorithm** A_{halt}, and returns "yes" if A_{halt} returns "yes", and returns "no" if A_{halt} returns "no".

On input M and w the algorithm $A_{\bar{u}}$ **does the following**.

1. It **builds a new Turing machine** M' that works as follows:
   - First M' writes w on its tape. (This may be done using $|w|$ states.)
   - Then M' moves back to the first letter of w and enters the initial state of M.
   - From there on, transitions of machine M are used.
   - For each halting but non-final state of M the machine M' goes into a new looping state that makes the machine move to the right indefinitely without ever halting.

   Clearly, this M' **halts on the blank initial tape if and only if** M **accepts the input** w.

2. Algorithm $A_{\bar{u}}$ does not execute M'. Instead, it just **gives** M' **as an input to the hypothetical algorithm** A_{halt}, and returns "yes" if A_{halt} returns "yes", and returns "no" if A_{halt} returns "no".

Algorithm $A_{\bar{u}}$ **works correctly:** it returns "yes" on input M and w if and only if M accepts w. But such algorithm cannot exist by the previous corollary. Hence, the algorithm A_{halt} cannot exist either.

A more careful analysis of the reduction above one concludes that the complement of L_{halt} is not recursively enumerable, *i.e.*, non-halting is not semi-decidable.

A more careful analysis of the reduction above one concludes that the complement of L_{halt} is not recursively enumerable, *i.e.*, non-halting is not semi-decidable.

We can also see this as follows. The language L_{halt} is recursively enumerable. (One can effectively simulate the given Turing machine on a blank tape until – if ever – it halts. The semi-algorithm answers "yes" if the simulation halts.)

Now, because L_{halt} is recursively enumerable but not recursive, its complement cannot be recursively enumerable.