Diagonalization language

All binary strings that are not accepted by the TM obtained by that string

The **diagonalization language** is the set

str$(M_i) = w_i$

$$L_d = \{w \mid w = w_i,\ w_i \notin L(M_i)\}$$

In words, L_d contains all binary strings w_i such that the i-th TM does not accept w_i

Diagonalization language

The following table reports whether M_i accepts (1) or rejects (0) w_j

$i \downarrow$ \ $j \rightarrow$	1	2	3	4	$\cdots$
1	0	1	1	0	$\cdots$
2	1	1	0	0	$\cdots$
3	0	0	1	1	$\cdots$
4	0	1	0	1	$\cdots$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$

Diagonalization language = strings that has 0 in the diagonal

Diagonal

Diagonalization language

We can interpret the i-th row of the table as the **characteristic vector** of language $L(M_i)$: an entry is 1 iff the corresponding string belongs to the language

Observation : The table represents the entire class RE. In fact, a language is in RE if and only if its characteristic vector is a row of the table

Diagonalization language

The following statements are logically equivalent

- the i-th element of the diagonal is 0
- $w_i \notin L(M_i)$
- $w_i \in L_d$

This means that, if we **complement** the diagonal, we obtain the characteristic vector of language L_d

This vector cannot be a row of the table, because the diagonal element of each row does not match with at least one position of the characteristic vector of language L_d → the intersection with the diagonal won't match → L_d is the complement of the diagonal

Diagonalization language

Theorem L_d is not in RE

Proof Let us assume that there is a TM M such that $L_d = L(M)$. Choose i such that $M_i = M$. Does the string w_i belong to L_d ?

If $w_i \in L_d$, then M_i accepts w_i because $L_d = L(M_i)$. But by definition of L_d, the i-th element of the diagonal is 0 and therefore M_i does not accept w_i

If $w_i \notin L_d$, then M_i does not accept w_i. But by definition of L_d, the i-th element of the diagonal is 1 and therefore M_i accepts w_i

We have therefore obtained a **contradiction** $\square$

Recursive languages

A language L is **recursive** (REC) if $L = L(M)$ for some TM M such that

- if $w \in L$, then M halts in a final state
- if $w \notin L$, then M halts in a non-final state

If we think of L as a decision problem P_L, then we say that P_L is **decidable** whenever L is recursive, and P_L is **undecidable** otherwise

Decidability corresponds to the notion of **algorithm**: we have a sequence of steps that always ends and produces some answer

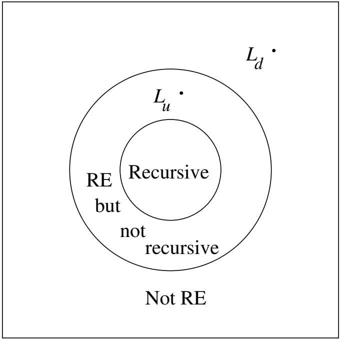
REC vs. RE$\setminus$REC

Comparison :

- recursive language means that there is an algorithm for **solving** the associated decision problem, that is, we always have an answer
- language in RE that is non-recursive means that we can **enumerate** the positive instances of the problem, but we cannot conclude in a finite amount of time that an instance has a negative answer

The distinction between decidable / undecidable problems is often more important than the distinction between RE / non-RE problems

Language classes

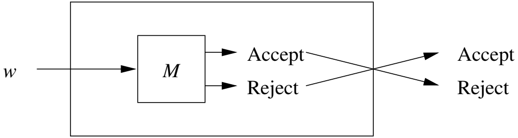


- recursive = decidable = M always halts
- RE = M halts upon acceptance
- non-RE = we cannot compute; **Example** : L_d

Properties of recursive languages

Theorem If L is recursive, then $\overline{L}$ is recursive

Proof If L is recursive, there is a TM M that always halts, such that $L(M) = L$. We construct a TM M' such that M' accepts when M does not, and *vice versa*. M' always halts and $L(M') = \overline{L}$ $\square$



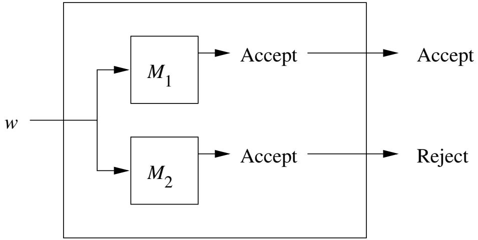
Corollary If L is in RE and $\overline{L}$ is not in RE, then L cannot be a recursive language

Properties of RE languages

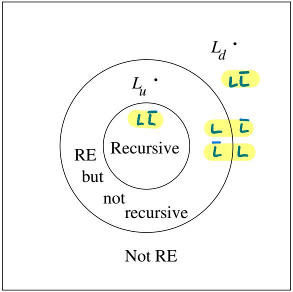
Theorem If L and $\overline{L}$ are in RE, then L is recursive

Proof Let $L = L(M_1)$ and $\overline{L} = L(M_2)$. We build a multi-tape TM M that simulates M_1 and M_2 in parallel

If the input is in L, M_1 accepts and halts, then also M accepts and halts. If the input is not in L, then M_2 accepts and halts, so M rejects and halts □



L and $\overline{L}$



Where can L and $\overline{L}$ be placed ?

Combinatorially, there are 9 possible arrangements, but the theory allows only 4 of them

L and $\overline{L}$

Possible arrangements for L and $\overline{L}$

- both L and $\overline{L}$ are recursive
- both L and $\overline{L}$ are not in RE
- L is RE but not recursive, and $\overline{L}$ is not RE
- $\overline{L}$ is RE but not recursive, and L is not RE

It is not possible that a language is recursive and the complement is RE but not recursive or not RE

It is not possible that a language and its complement are both RE but not recursive

Example

Let us consider the language $\overline{L_d}$, which contains the strings w_i such that M_i accepts w_i

Since L_d is not RE, $\overline{L_d}$ is not recursive. It is possible that $\overline{L_d}$ is not RE, or alternatively RE but not recursive

We will prove later that $\overline{L_d}$ is RE but not recursive

Universal language

We want to encode pairs (M, w) consisting of

- one TM M with binary input alphabet
- one binary string w

We use $\text{enc}(M)$ followed by 111, followed by w, and write $\text{enc}(M, w)$.
Note : the sequence 111 never appears in $\text{enc}(M)$

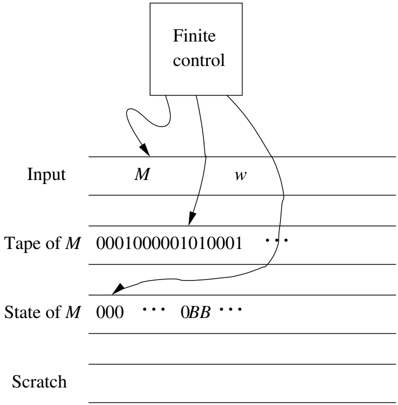
The language L_u, called **universal language**, is the set

$$L_u = \{\text{enc}(M, w) \mid w \in L(M)\}$$

In words, L_u is the set of binary strings that encode a pair (M, w) such that $w \in L(M)$

Universal TM

There exists a TM U, called **universal** TM, such that $L(U) = L_u$



Universal TM

U (multi-tape version) has four tapes

- tape 1 contains the input string enc(M, w)
- tape 2 simulates M's tape, using the 0^j format for each X_j tape symbol, and 1 as cell separator
- tape 3 records M's state, using the 0^j format for each state q_j
- tape 4 : auxiliary copying tape, used to "enlarge" or "shrink" the available space for the 0^j representations in tape 2

Universal TM

Strategy exploited by U

- if $\text{enc}(M)$ is invalid, U halts and rejects (in this case $L(M) = \emptyset$)
- write w on tape 2 using 1 as separator, 0^1 for $0 = X_1$, and 0^2 for $1 = X_2$

  No encoding for B, use U's blank
- write the initial state on tape 3, using 0 for q_1, and place the tape head of tape 2 on the first cell
- search on tape 1 for a transition of the form $0^i10^j10^k10^l10^m$, where
  - 0^i is the state on tape 3
  - 0^j is M's tape symbol under the tape head of tape 2

Universal TM

Strategy exploited by U (cont'd)

- in order to simulate trasition $0^i 1 0^j 1 0^k 1 0^l 1 0^m$, the TM U
  - replaces the content of tape 3 with 0^k (new state)
  - replaces 0^j on tape 2 with 0^l (new tape symbol); if needed, we can "enlarge" or "shrink" U's tapes using the auxiliary tape (tape 4)
  - move the tape head of tape 2 to the left if $m = 1$ or to the right if $m = 2$, until the next 1 is reached (separator)
- if there is no transition $0^i 1 0^j 1 0^k 1 0^l 1 0^m$, M halts and U halts as well
- if M reaches a final state, then U halts and accepts

Universal language

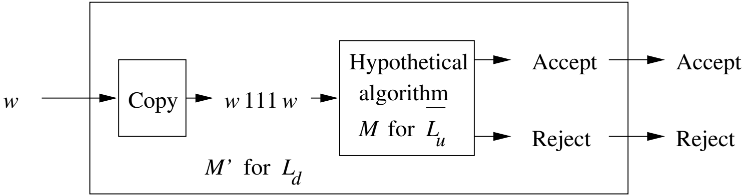
Theorem L_u is in RE but is not recursive

Proof L_u is in RE, since we have built the TM U

Let us assume that L_u is recursive. Then $\overline{L_u}$ is also recursive

Let M be a TM such that $L(M) = \overline{L_u}$. We build a new TM M' for L_d as follows (example of a **reduction**, a notion which we will introduce in the next section)

Universal language



On input $w = w_i$, M' builds $\text{enc}(M_i, w_i) = w_i 111 w_i$

M always halts, and accepts if and only if $w_i \notin L(M_i)$. As a consequence, M' always halts, and $L(M') = L_d$

We have a **contradiction**, since L_d is not recursive □

The halting problem

Given a TM M, we define $H(M)$ the set of strings w such that M **halts** with input w

Let us consider the language L_h, called the **halting problem**

$$L_h = \{\text{enc}(M, w) \mid w \in H(M)\}$$

There exists a TM M such that $L(M) = L_h$: M takes as input a pair enc(M', w) and simulates a computation of M' on w

M accepts whenever M' halts on w

Therefore L_h is a RE language

The halting problem

We can prove that L_h is not recursive (proof omitted)

Hence there is **no algorithm** that can state whether a given program ends or not on a given input

However, there exists a procedure that

- halts, if a given program ends on a given input
- cycles, if a given program does not end on a given input