Decidability and Undecidability

Classification of languages/problems

1) recursive languages: class of languages accepted by T.M.s that always halt

T.M that always halts = algorithm = effective procedure
(halts on all inputs in finite time, either accepting or rejecting)

<-> decidable problems/languages

problems/languages that are non recursive are called undecidable
=> they don't have algorithms

Note: regular and context-free languages are special cases of recursive languages

2) recursively enumerable (R.E.) languages:
class of languages defined by T.M. (or procedures)
arbitrary T.M. (that may not halt) = procedure

3) non-R.E. languages
languages/problems for which there is no T.M./procedure

Pictorially

1) algorithm (recursive)

$$\begin{tikzcd} \text{input } w \arrow[r] & \boxed{A} \arrow[r] \arrow[dr] & \text{yes } (w \in L) \\ & & \text{no } (w \notin L) \end{tikzcd}$$

2) procedure (R.E.)

$$\begin{tikzcd} \text{input } w \arrow[r] & \boxed{P} \arrow[r] & \text{yes } (w \in L) \end{tikzcd}$$

3) non-R.E. input $w \longrightarrow$??

The Church-Turing Thesis

We have provided a classification of languages according to TM-acceptance/termination.

However, we are interested in problems and their computability.

Decision problem: is a set of related questions with a yes/no answer

E.g. the decision problem of determining whether a natural number is prime is the set of questions

p_0 : is 0 prime?
p_1 : is 1 prime?
p_2 : is 2 prime?
⋮

Each question is an instance of the problem $<$ positive instances / negative instances

A solution to a decision problem P is an algorithm that determines the correct answer to each question $p_i \in P$.

P is decidable if it has a solution.

What is an algorithm (or effective procedure)?
difficult to define, but we can list some fundamental properties:

- complete: produces correct answer for each instance
- mechanistic: consists of a finite sequence of instructions, each of which can be carried out unambiguously
- deterministic: performs the same computation if executed multiple times on the same input

TM machine computations are mechanistic and deterministic.
If the TM halts on all inputs, then it is an algorithm.

Representation of decision problems:

To provide a TM solution to a decision problem, we need to encode its instances in terms of strings

$$\begin{tikzcd} \text{problem instance} \arrow[r, "\text{representation}"] & \text{TM input} \arrow[r, "\text{TM computation}"] & \text{yes/no answer} \\ P_i \arrow[r] & w_i \arrow[r] & \text{yes/no} \end{tikzcd}$$

Note: The choice of the representation may affect the solution. However, it will not affect existence of a TM that is an effective procedure.

28/10/2013

Would it make a difference, if we choose a different computation model than the TM?

Several such models have been proposed, based on different paradigms:

- string transformations:
  - Post systems [Emil Post, 1936]
  - Semi-Thue systems [Axel Thue, 1914]
  - Markov Systems [Markov, 1961]
  - unrestricted grammars [Noam Chomsky, 1956]
- function evaluation:
  - partial recursive and μ-recursive functions [Gödel 1931, Kleene 1936]
  - lambda calculus [Church 1941]
- abstract computing machines:
  - Turing Machines
  - register machines [Stephenson 1960]
  - multi-stack machines

- programming languages
  - while programs [Kfoury et al. 1982]

It has been shown that all of the above algorithmic systems have the same computing power, i.e., they can simulate each other.

⇒ It is currently believed that the computing power of any of these systems defines the intrinsic limits of computation.

This belief is known as Church-Turing Thesis

<u>Church-Turing Thesis for decision problems:</u>

There is an effective procedure to solve a decision problem iff there is a TM that always halts and solves the problem

A partial solution to a decision problem P is a not necessarily complete but otherwise effective procedure that returns yes for every positive instance $p \in P$.

If p is a negative instance, the procedure may return no or not terminate.

<u>Church-Turing Thesis for recognition problems:</u>

A decision problem is partially solvable iff there is a TM that accepts the positive instances of the problem

The most general formulation is in terms of TMs that compute functions (by leaving the result on the tape)

<u>Church-Turing Thesis for computable functions:</u>

A function f is effectively computable iff there is a TM that computes f.

Note : The Church-Turing thesis cannot be proved, since it relies on the intuitive notion of effective procedure

It could be disproved, by exhibiting an effective procedure not computable by a TM. This is unlikely, given the robustness of the TM model

We exploit the Church-Turing thesis to simplify proofs of existence of a decision algorithm:

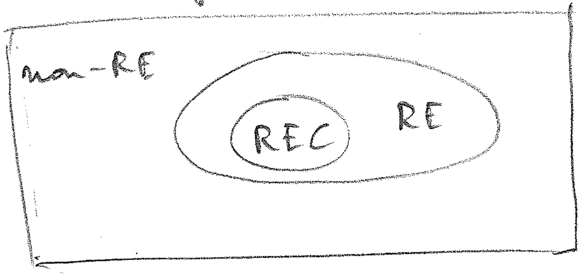
instead of constructing the TM we describe an intuitively effective procedure to solve the problem.

Classes of languages:

REC ... class of recursive languages

RE ... class of recursive enumerable languages

non-RE .. class of non recursive enumerable languages

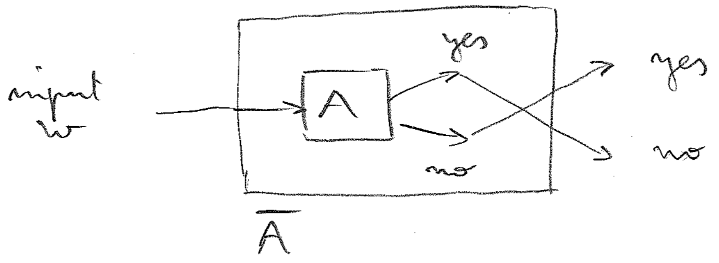


Closure properties

Theorem: L is recursive $\Longrightarrow$ $\overline{L}$ is recursive

Proof: given algorithm A for L
construct alg. $\overline{A}$ for $\overline{L}$

Intuitively:



More precisely: $A = (Q, \Sigma, \Gamma, \delta, q_0, \not{b}, F)$

$$\overline{A} = (Q \cup \{\bar{f}\}, \Sigma, \Gamma, \bar{\delta}, q_0, \not{b}, \{\bar{f}\})$$

yes $\longrightarrow$ no : we assume that final states of F are blocking.
So we get that $\overline{A}$ blocks in a non-final state

no $\longrightarrow$ yes : if $\delta(q, x)$ is undefined
then set $\bar{\delta}(q, x) = (\bar{f}, x, R)$

Since L is recursive, A always halts (with output yes or no),
hence $\overline{A}$ also halts always (with output no or yes)

q.e.d.

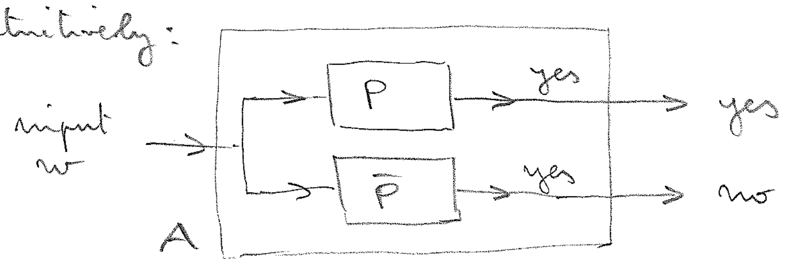
<u>Theorem</u>: Both L and $\bar{L}$ are R.E.

$\Rightarrow$ both L and $\bar{L}$ are recursive

Proof: Let P, $\bar{P}$ be procedures (i.e. TMs) for L, $\bar{L}$.

We run them in parallel, to get an alg. A for L.

Intuitively:



Note: we assume that $\bar{P}$ is non-blocking on non-final states. (What does this mean from a formal point of view?)

Note: for every w, one of P, $\bar{P}$ will halt and give the right answer.

More precisely:

For A use 2 tapes, one simulating that of P
one — " — $\bar{P}$

Initially: copy input string w to tape 2.

Further states of A: for each state p of P, for each state q of $\bar{P}$ } a state $\langle p,q \rangle$ of A

Transitions:

for each pair of transitions: $\begin{cases} \delta(p,x) = (p',x',d_1) & \text{in } P \\ \delta(q,y) = (q',y',d_2) & \text{in } \bar{P} \end{cases}$

we have a transition in A

$$\delta(\langle p,q \rangle, x, y) = (\langle p',q' \rangle, (x',d_1), (y',d_2))$$

Final states: every $\langle p,q \rangle$ s.t. p is final in P

Note: if $\bar{P}$ accepts in q, then q is final (and we assume it is halting). Hence, if A reaches $\langle p,q \rangle$, p cannot be final, and since q is halting, A rejects.

q.e.d.

The two previous results imply that, for every language L, we have that

- either both L and $\bar{L}$ are recursive
- or at least one of L, $\bar{L}$ is non-R.E.

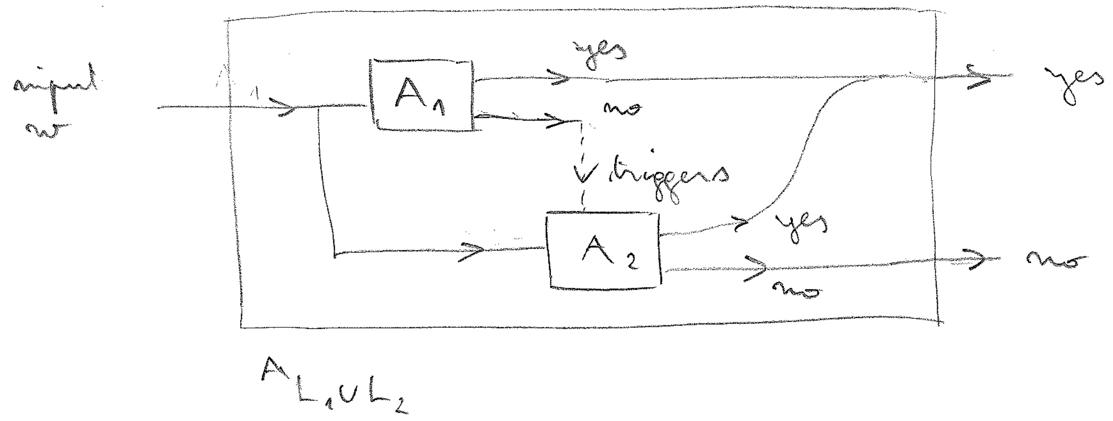
	$\bar{L}$ rec.	$\bar{L}$ R.E. but not rec.	$\bar{L}$ non-R.E.
L rec.	✓	×	×
L R.E. but not rec.	×	×	✓
L non-R.E.	×	✓	✓

(× … means that this case is not possible)

<u>Theorem</u>: L_1, L_2 rec. $\Rightarrow$ $L_1 \cup L_2$ rec.

(recursive languages are closed under union)

Proof: let A_1, A_2 be algorithms for L_1, L_2



Output "no" of A_1 triggers A_2 means:
if A_1 halts in a non-final state q (i.e. $w \notin L_1$), then we have a transition from q to the initial state of A_2 (to feed w to A_2, we can store it on a second tape before running A_1)

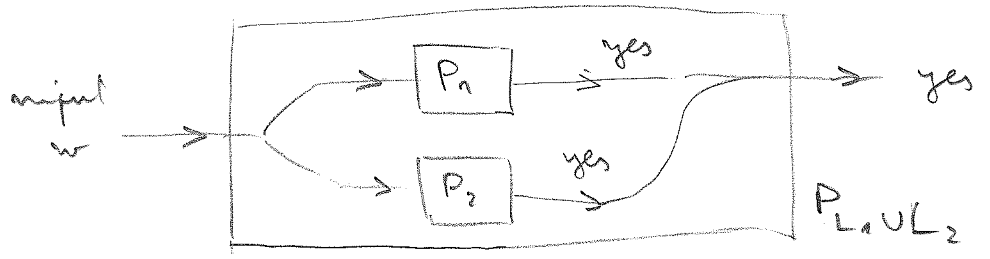
$\boxed{\text{Exercise:}}$ Provide the formal details of the construction of $A_{L_1 \cup L_2}$ given A_1 and A_2. q.e.d.

Theorem: L_1, L_2 R.E. $\Longrightarrow$ $L_1 \cup L_2$ R.E.

(R.E. languages are closed under union)

Proof.: let P_1, P_2 be procedures (i.e. TMs) for L_1, L_2

we run P_1, P_2 in parallel



Note: we assume that P_1, P_2 are non-blocking on non-final states

Note: if $w \in L_1 \cup L_2$, one of P_1 or P_2 will halt and answer yes

Exercise: work out the details

Exercise: prove / disprove closure under intersection, reversal

28/10/2014

Showing languages to be undecidable / non-R.E.

To show languages to be undecidable / non-R.E. we make use of the basic idea of feeding the encoding of a T.M. as input to a T.M.

e.g. • Universal T.M. (UTM):

- input: T.M. M
  M's input string w } $\langle M, w \rangle$

- output: UTM accepts $\langle M, w \rangle$ $\Longleftrightarrow$ M accepts w

• Diagonalization: consider what happens when certain T.M.s are fed their own encoding as input.

In both cases we need a suitable way of encoding T.M.s by means of strings

We consider <u>encoding T.M.s</u> by means of strings over $\{0,1\}$, i.e., <u>as binary integers.</u>

Let $M = (Q, \Sigma, \Gamma, \delta, q_1, ⊭, F)$ be a T.M.

We assume w.l.o.g. $\Sigma = \{0, 1\}$, $\Gamma = \{0, 1, ⊭\}$ } i.e., we consider only T.M.s over the binary alphabet (this is no limitation)

$Q = \{q_1, q_2, \dots, q_r\}$

- initial state q_1
- single final state q_2

We use the following notation:

$x_1 = 0$, $x_2 = 1$, $x_3 = ⊭$

$d_1 = \text{left}$, $d_2 = \text{right}$, $d_3 = \text{stay}$

Encoding of transitions: $\delta(q_i, x_j) = (q_k, x_l, d_m)$

with $i, k \in \{1, \dots, r\}$

$j, l \in \{1, 2, 3\}$

$m \in \{1, 2, 3\}$

We encode the transition as $0^i 1 0^j 1 0^k 1 0^l 1 0^m$

Encoding $\mathcal{E}(M)$ of entire T.M. M.

Let $c_1, \dots, c_n$ be the encodings of the transitions of M.

We encode M as:

$$\mathcal{E}(M) = c_1 11 c_2 11 c_3 11 \cdots 11 c_n 11$$

Encoding of M with its input w: $\mathcal{E}(M)\, 111\, w = \langle M, w\rangle$

(note: the first 111 indicate that the encoding $\mathcal{E}(M)$ is finished)

Note:

- each bit string encodes a unique T.M.:
  - either it is a valid encoding and encodes a unique T.M.
  - or it is not a valid encoding according to our rules; in this case we assume that it encodes the particular machine M_0 with 1 state and no transitions ($\mathcal{L}(M_0) = \emptyset$)
- each T.M. admits at least 1 encoding (possibly many)

<u>Enumerating binary strings</u>:

We define an ordering on binary strings:

- in increasing order of length
- strings of the same length are ordered lexicographically

$\Rightarrow \varepsilon, 0, 1, 00, 01, 10, 11, 000, 001, \dots$

Let w_i be the i-th string in this ordering (starting with $w_1 = \varepsilon$)

We define M_i as the T.M. encoded by w_i, i.e. $w_i = \mathcal{E}(M_i)$

$\Rightarrow$ we get an <u>ordering of T.M.s</u>:

- each T.M. appears at least once in the ordering
- —"— may appear many times

Note: each binary string w_i can be viewed:

- as a string fed as input to a TM
- as the encoding $w_i = \mathcal{E}(M_i)$ of a TM M_i

The diagonalization language

Exploiting the ordering/enumeration of w_i / M_i, we can consider the infinite table T s.t. $\forall i, j \geq 1$:

$$T(i,j) = \begin{cases} 1 & \text{if } w_j \in \mathcal{L}(M_i) \\ 0 & \text{if } w_j \notin \mathcal{L}(M_i) \end{cases}$$

	w_1	w_2	w_3	w_4	$\dots$
M_1	0	1	1	0	
M_2	1	1	0	1	
M_3	0	0	1	1	
M_4	1	0	1	0	
$\vdots$					$\ddots$

Each row of T is a characteristic vector of $\mathcal{L}(M_i)$, specifying which strings belong to $\mathcal{L}(M_i)$.

Definition: The diagonalization language

$$L_d = \{ w_i \mid T(i,i) = 0 \} = \{ w_i \mid w_i \notin \mathcal{L}(M_i) \}$$

In other words: L_d is defined as the language whose characteristic vector is the bit by bit complementation of the diagonal of T.

Theorem: L_d is non-R.E.

Proof: By contradiction, assume L_d is R.E. and has a T.M. that accepts it.

Then $\exists k \geq 1$ s.t. $\mathcal{L}(M_k) = L_d$

Question : is $w_k \in L_d$

Case 1: $w_k \in L_d \Rightarrow w_k \in \mathcal{L}(M_k)$

$\Rightarrow T(k,k) = 1$

$\Rightarrow w_k \notin L_d$ contradiction

Case 2: $w_k \notin L_d \Rightarrow w_k \notin \mathcal{L}(M_k)$

$\Rightarrow T(k,k) = 0$

$\Rightarrow w_k \in L_d$ contradiction

Intuition: L_d is defined so that it disagrees with each $\mathcal{L}(M_i)$ on at least string w_i.

$\Rightarrow$ no M_i can have L_d as its language

But all T.M. s appear in the enumeration

$\Rightarrow$ no T.M. can accept L_d

<u>Universal T.M.</u>

UTM: Input $\langle M, w \rangle$ with - $\mathcal{E}(M)$ encoding of a T.M. M
i.e. $\mathcal{E}(M)111w$ - w input string for M

Action: UTM simulates M on w, and accepts if and only if M accepts w.

Language L_u of UTM

Definition : Universal language

$$L_u = \{ \langle M, w \rangle \mid w \in \mathcal{L}(M) \}$$

Note: we use $\langle M, w \rangle$ to denote $\mathcal{E}(M)111w$, where $\mathcal{E}(M)$ denotes the encoding of T.M. M, as introduced previously.

<u>Theorem</u>: L_u is R.E.

Proof: we construct a T.M. U s.t. $\mathcal{L}(U) = L_u$

U has 4 tapes:

tape 1: input tape containing $\langle M, w\rangle$ (read-only)

tape 2: simulates the tape of M

tape 3: contains the current state q_i of M: $\underbrace{00\cdots0}_{i}$

(note: the state of M cannot be encoded in the state of U, since we have no bound on the number of states that M could have)

tape 4: scratch tape

Transitions: U simulates the transitions stored on tape 1 by modifying tapes 2 and 3

- initially, copy w to tape 2, and $q_1 = 0$ to tape 3
  initial state
- to simulate each transition of M, U uses
  - the current state $q_i = 0^i$ on tape 3
  - the current symbol X_j on tape 2

  looks on tape 1 for transition $0^i 1 0^j 1 0^k 1 0^l 1 0^m$,
  i.e. $\delta(q_i, X_j) = (q_k, X_l, d_m)$

  and
  - changes the content of tape 3 to $q_k = 0^k$
  - changes the current symbol on tape 2 to X_l
  - moves the head on tape 2 according to d_m

U accepts whenever M enters final state q_2

To show that L_u is not recursive, we exploit the notion of reduction:

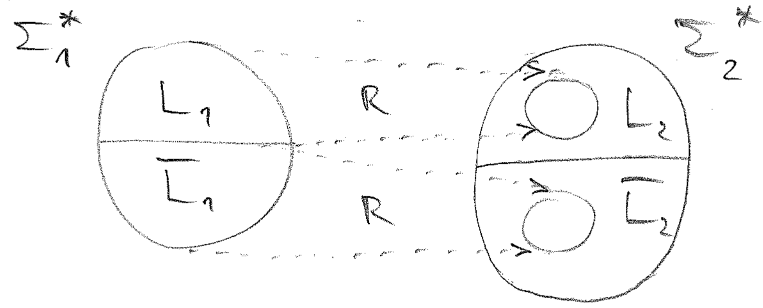
Reduction

28/10/2014
(afternoon)

<u>Definition</u>: L_1 reduces to L_2 (denoted $L_1 < L_2$) if there exist a function R (called the reduction from L_1 to L_2) such that:

1) R is computed by some T.M. M_R that takes as input a string w (over the alphabet Σ_1 of L_1) and <u>halts</u> leaving on its tape a string $R(w)$ (over the alphabet Σ_2 of L_2)
   Note: M_R is an algorithm
2) $w \in L_1 \iff R(w) \in L_2$

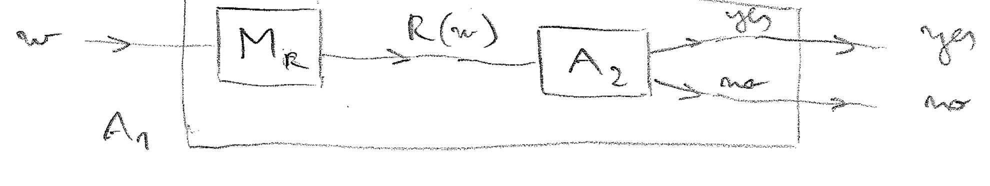
Intuitively:



R maps: all strings in L_1 to a subset of all strings in L_2
— " — $\overline{L_1}$ — " — $\overline{L_2}$

<u>Theorem</u>: $L_1 < L_2$ and L_2 is recursive $\Rightarrow$ L_1 is recursive

Proof: given algorithm A_2 for L_2
— " — M_R for R
construct algorithm A_1 for L_1: (A_1 is correct since $w \in L_1 \iff R(w) \in L_2$)



q.e.d.

We can use the same result to show a language to be non-recursive (i.e., undecidable):

Corollary: $L_1 < L_2$ and L_1 is non-recursive

$\Rightarrow$ L_2 is non-recursive

The above results apply also to R.E.

Theorem: $L_1 < L_2$ and L_2 is R.E. $\Rightarrow$ L_1 is R.E.

$L_1 < L_2$ and L_1 is non-R.E. $\Rightarrow$ L_2 is non-R.E.

Intuitively: $L_1 \leq L_2$ means that L_2 is at least as difficult as L_1.

Theorem: L_u is non-recursive

We show that $\overline{L}_d < L_u$ (i.e., $\overline{L}_d$ reduces to L_u).

The claim follows, since $\overline{L}_d$ is non-recursive

(if $\overline{L}_d$ were recursive, also $\overline{\overline{L}}_d = L_d$ would be recursive, but we know that L_d is non-R.E.)

Reduction R: given input string w for $\overline{L}_d$

produce input string $\langle M, w\rangle = \mathcal{E}(M)111w$ for L_u

We define $R(w) = w\,111\,w$.

Clearly, there exists an algorithm M_R to convert w into $w111w$

We need to show: $w \in \overline{L}_d \Leftrightarrow R(w) \in L_u$

$w_i \in \overline{L}_d \Leftrightarrow w_i \in \mathcal{L}(M_i) \Leftrightarrow \langle M_i, w_i\rangle \in L_u$

$\Leftrightarrow \mathcal{E}(M_i)111w_i \in L_u \Leftrightarrow w_i\,111\,w_i \in L_u$

$\Leftrightarrow R(w_i) \in L_u$ q.e.d.

To sum up:

L_d is non-R.E. : direct proof via diagonalization

$\overline{L}_d$ is R.E., but non-recursive : exercise

L_u is R.E., but non-recursive : R.E. by construction of U
non-rec. by $\overline{L}_d < L_u$

$\overline{L}_u$ is non-R.E. : by inference from previous line

We can exploit this to show a language L to be non-rec. or non-R.E.

- $\overline{L}_d < L$ or $L_u < L$ $\Rightarrow$ L is non-recursive
  $L_d < L$ or $\overline{L}_u < L$ $\Rightarrow$ L is non-R.E.
  (and hence non-recursive)

Consider the following languages over $\Sigma = \{0,1\}$

$L_e = \{ \mathcal{E}(M) \mid \mathcal{L}(M) = \emptyset \}$

$L_{ne} = \{ \mathcal{E}(M) \mid \mathcal{L}(M) \neq \emptyset \}$

Hence: L_e ... set of all strings that encode T.M.s that accept the empty language
L_{ne} ... complement of L_e

We have that: L_{ne} is R.E. but non-recursive
L_e is non-R.E.

Proof: see Exercise 5.

We have shown that a specific property of T.M. languages (namely non-emptiness) is undecidable

This is just a special case of a much more general result:
All non-trivial properties of R.E. languages are undecidable.

Property P: of R.E. languages is a set of R.E. languages

e.g. the property of being context-free is the set of all CFLs.
the — · · — empty is the set $\{\emptyset\}$ consisting of only $\emptyset$

A property is trivial if either all or no R.E. language has it.
⇒ P is non-trivial if at least one R.E. language has P
and — · · — does not have P

Note: a T.M. cannot recognize a property (i.e. a set of languages) by taking as an input string a language, because a language is typically infinite

⇒ we consider instead a property P as the language of the codes of those T.M.s that accept a language that satisfies P

$$L_P = \{ \mathcal{E}(M) \mid \mathcal{L}(M) \text{ has property } P \}$$

Rice's Theorem: every non-trivial property of R.E. languages is undecidable

Proof: let P be a non-trivial property of R.E. languages
assume $\emptyset$ does not have P (otherwise, we can work with $\overline{P}$)

Since $\mathcal{P}$ is non-trivial, there is some $L_1 \in \mathcal{P}$ with $L_1 \neq \emptyset$. Let M_1 be s.t. $\mathcal{L}(M_1) = L_1 \Rightarrow \mathcal{E}(M_1) \in L_{\mathcal{P}}$

We show that $L_u < L_{\mathcal{P}}$:

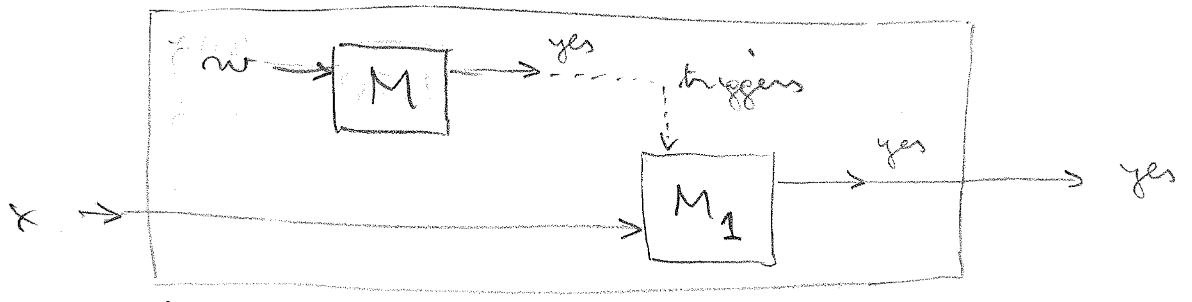
Reduction R is an algorithm that:

$$\begin{tikzcd}
\mathcal{E}(M) \arrow[r] & \boxed{R} \arrow[r] & \mathcal{E}(M_2) \\
w \arrow[ur] & &
\end{tikzcd}$$

- takes as input a pair $\langle M, w \rangle$ considered as instance of L_u
- produces a code $\mathcal{E}(M_2)$ for a TM M_2

s.t. $\langle M, w \rangle \in L_u \Leftrightarrow \mathcal{E}(M_2) \in L_{\mathcal{P}}$

Idea for M_2: with $\mathcal{E}(M_2) = R(\langle M, w \rangle)$



M_2

- M_2 ignores first its own input and writes w on tape 2
- simulates M on w (on tape 2),
- if M accepts w: M_2 starts simulating M_1 on x and accepts if M_1 accepts x

if M rejects w or does not halt, M_2 does the same

Note: since R takes as input $\langle M, w \rangle$, it can hardcode $\mathcal{E}(M)$ and w into M_2

We get that: $w \in \mathcal{L}(M) \Rightarrow \mathcal{L}(M_2) = \mathcal{L}(M_1) \Rightarrow \mathcal{E}(M_2) \in L_{\mathcal{P}}$

$w \notin \mathcal{L}(M) \Rightarrow \mathcal{L}(M_2) = \emptyset \Rightarrow \mathcal{E}(M_2) \notin L_{\mathcal{P}}$

$\Rightarrow \langle M, w \rangle \in L_u \Leftrightarrow w \in \mathcal{L}(M) \Leftrightarrow \mathcal{E}(M_2) \in L_{\mathcal{P}}$

$\Rightarrow$ R reduces L_u to $L_{\mathcal{P}}$ $\Rightarrow L_{\mathcal{P}}$ is undecidable q.e.d.