10/1/2005 (3.1)

$\boxed{21/12/2005}$

Decidability and Undecidability

Classes of languages/problems:

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

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

$\leftrightarrow$ decidable problems/languages

problems/languages that are non recursive are called undecidable
$\Rightarrow$ 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

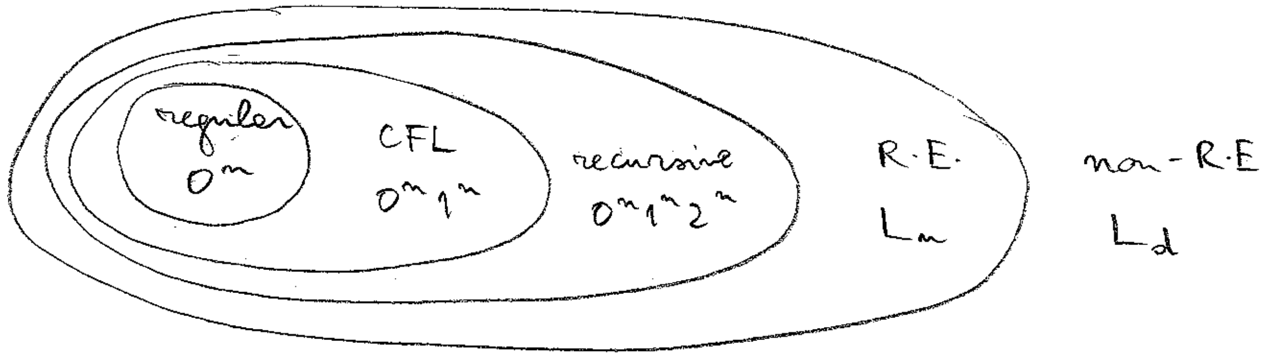
Pictorially

1) algorithm (recursive) — input $w \longrightarrow \boxed{A}$ $\rightarrow$ yes $(w \in L)$; $\rightarrow$ no $(w \notin L)$

2) procedure (R.E.) — input $w \longrightarrow \boxed{P} \longrightarrow$ yes $(w \in L)$

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

To sum up

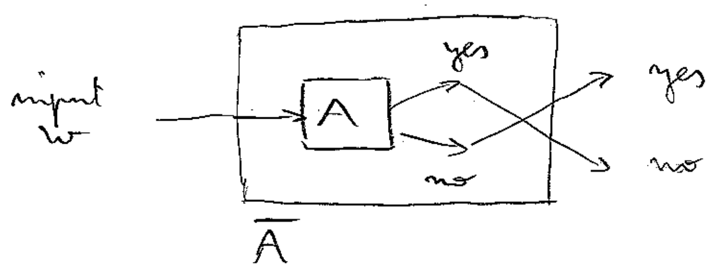


Closure properties

Theorem: L is recursive $\Rightarrow$ $\bar{L}$ is recursive

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

Intuitively:



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

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

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

no $\rightarrow$ 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 $\bar{A}$ also halts always (with output no or yes)

q.e.d.

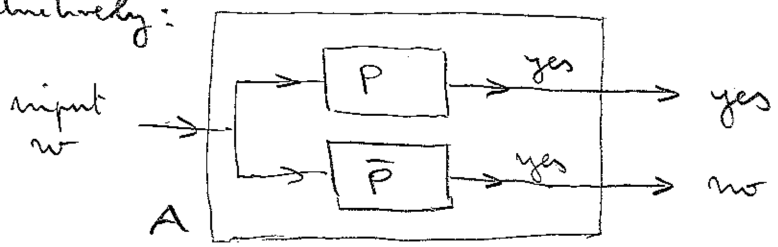
Theorem: Both L and $\bar{L}$ are R.E.

$\Rightarrow$ 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

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 — " — $\bar{P}$

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

Transitions:

for each transition $\delta(p, x) = (p', x', D_1)$ in P
— " — $\delta(q', y) = (q', y', D_2)$ in $\bar{P}$

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 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.

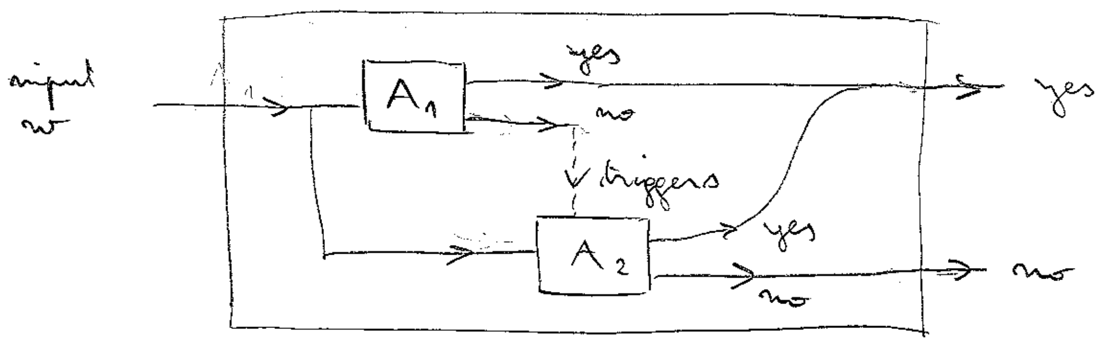
	$\bar{L}$ rec.	$\bar{L}$ R.E. but not rec.	$\bar{L}$ non-R.E.
L rec.	✓	✗	✗
L R.E. but not rec.	✗	✗	✓
L non-R.E.	✗	✓	✓

(✗ ... means that this case is not possible)

Theorem: 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



$A_{L_1 \cup L_2}$

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)

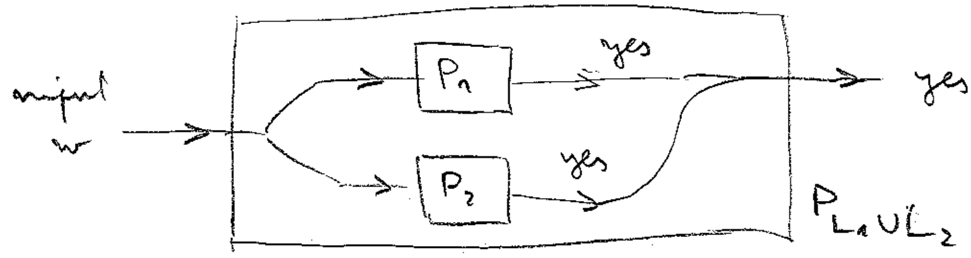
q.e.d

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

$\boxed{\text{Exercise:}}$ work out the details

$\boxed{\text{Exercise}}$ prove / disprove closure under intersection, reversal

$\boxed{9/1/2006}$

<u>Showing languages to be undecidable / non-R.E.</u>

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 $\Big\}$ $\langle M, w \rangle$

- output: UTM accepts $\langle M, w \rangle \iff$ 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 encoding T.M.s by means of strings over $\{0, 1\}$, i.e., as binary integers.

Let $M = (Q, \Sigma, \Gamma, \delta, q_1, \not b, F)$ be a T.M.

We assume w.l.o.g.
$\Sigma = \{0, 1\}$,
$\Gamma = \{0, 1, \not b\}$
} i.e., we consider only T.M.s over the binary alphabet (this is no limitation)

$Q = \{q_1, q_2, \ldots, q_r\}$
- initial state q_1
- single final state q_2

We use the following notation:

$x_1 = 0, \quad x_2 = 1, \quad x_3 = \not b$

$d_1 = \text{left}, \quad d_2 = \text{right}$

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

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

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

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

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 $c_1, \ldots, c_n$ 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: $\mathcal{E}(M)\ 111\ w$

(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

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

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	...
M_1	0	1	1	0	
M_2	1	1	0	1	
M_3	0	0	1	1	
M_4	1	0	1	0	
$\vdots$					

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

<u>Definition</u>: 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.

<u>Theorem</u>: 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

Universal T.M.

UTM: Input $\langle \mathcal{E}(M), w \rangle$ with $\mathcal{E}(M)$: encoding of a T.M. M

– w input string for M

Action: UTM simulates M on w, and accepts $\langle \mathcal{E}(M), w \rangle$ if and only if M accepts w.

Language L_u of UTM

Definition: Universal language

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

Theorem 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 $\langle \mathcal{E}(M), w \rangle$ (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 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

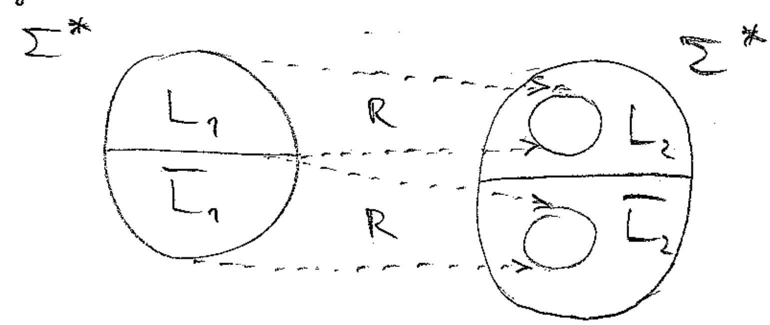
To show that L_u is not recursive, we exploit the notion of reduction:

Reduction

Definition: 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 (an instance of L_1) and halts leaving a string $R(w)$ on its tape (M_R is an algorithm!)
2) $w \in L_1 \iff R(w) \in L_2$

Intuitively:

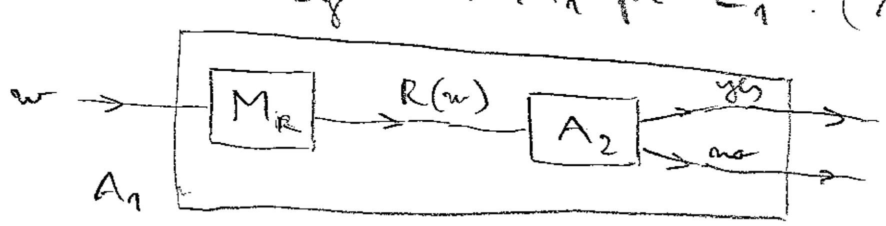


R maps: all strings in L_1 to a subset of all strings in L_2
— " — $\overline{L_1}$ — " — $\overline{L_2}$

Theorem: $L_1 < L_2$ 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):

11/1/2006

<u>Corollary:</u> $L_1 < L_2$ 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.

<u>Theorem:</u> 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 $\langle \mathcal{E}(M), w \rangle$ for L_u

We define $R(w) = \langle w, w \rangle = w\,111\,w$

Clearly, there exists an algorithm M_R to convert w into $\langle w, w \rangle$

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

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

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 10 (10.1, 10.2)

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.
$\Rightarrow$ 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

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

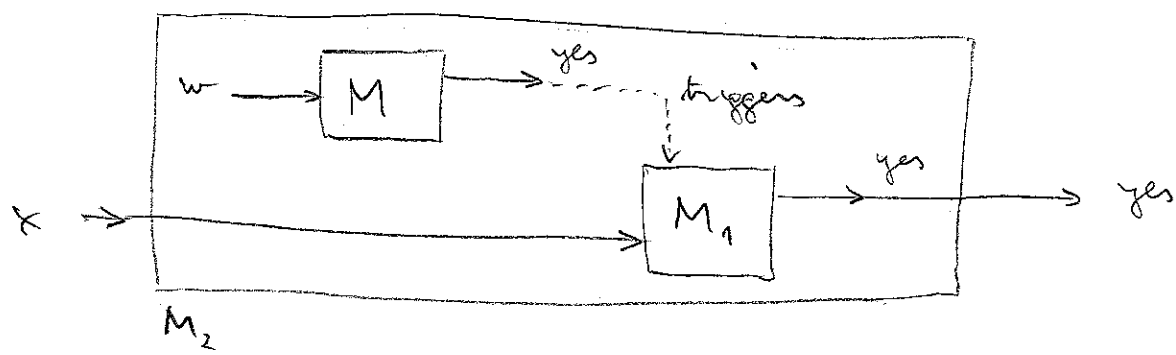
We show that $L_u < L_P$:

Reduction R is an algorithm that:

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

s.t. $\langle \mathcal{E}(M), w \rangle \in L_u \iff \mathcal{E}(M_2) \in L_P$

Idea for M_2:



- M_2 ignores first its own input x, and writes w on tape 2
- simulates M on w using an UTM (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 $\langle \mathcal{E}(M), w \rangle$, it can hardcode w into M_2

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

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

$\Rightarrow \langle \mathcal{E}(M), w \rangle \in L_u \iff w \in \mathcal{L}(M) \iff \mathcal{E}(M_2) \in L_P$

$\Rightarrow$ R reduces L_u to L_P $\Rightarrow$ L_P is undecidable q.e.d.