

# COSE215: Theory of Computation

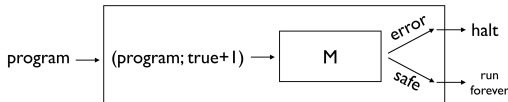
## Lecture 16 — Undecidability

Hakjoo Oh  
2017 Spring

# Undecidable Problems

Informally, undecidable problems are those that cannot be solved by computers (no algorithms exist). Examples:

- Halting problem
  - ▶ See <https://www.youtube.com/watch?v=92WHN-pAFCs>
- Reasoning about programs



- Properties of CFG:
  - ▶ Is a given CFG ambiguous?
  - ▶ For CFGs  $G_1$  and  $G_2$ , is  $L(G_1) \cap L(G_2) = \emptyset$ ?
  - ▶ Is  $L(G_1) = L(G_2)$ ?
  - ▶ Is  $L(G_1) = L(R)$  for some regular expression  $R$ ?
  - ▶ Is  $L(G_1) = T^*$  for some alphabet  $T$ ?
- Post's Correspondence Problem (PCP)

## Post's Correspondence Problem

We can describe this problem as a type of puzzle. We are given a collection of dominos, each containing two strings, one on each side: e.g.,

$$\left\{ \left[ \frac{b}{ca} \right], \left[ \frac{a}{ab} \right], \left[ \frac{ca}{a} \right], \left[ \frac{abc}{c} \right] \right\}$$

The task is to make a list of dominos (repetitions permitted) so that the string we get by reading the symbols on the top is the same as the string of symbols on the bottom. This list is called a *match*: e.g.,

$$\left[ \frac{a}{ab} \right] \left[ \frac{b}{ca} \right] \left[ \frac{ca}{a} \right] \left[ \frac{a}{ab} \right] \left[ \frac{abc}{c} \right]$$

For some collection of dominos, finding a match may not be possible: e.g.,

$$\left\{ \left[ \frac{abc}{ab} \right], \left[ \frac{ca}{a} \right], \left[ \frac{acc}{ba} \right] \right\}$$

Post's Correspondence Problem is to determine whether a collection of dominos has a match.

# Recursively Enumerable Languages

## Definition

A language  $L$  is *recursively enumerable* (RE) if there exists a Turing machine that accepts it.

$$L \text{ is RE} \Leftrightarrow \exists M \in TM. \forall w \in L. q_0 w \vdash^* x_1 q_f x_2$$

# Recursive Languages (Decidable Languages)

## Definition

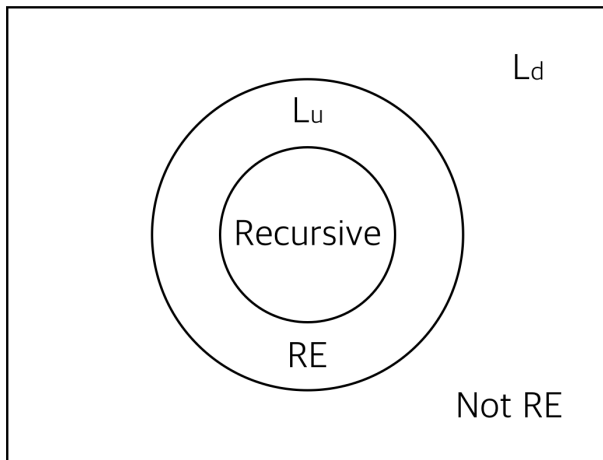
A language  $L$  is *recursive* if there exists a Turing machine that accepts it and always terminates.

In other words, we call a language  $L$  *recursive* if  $L = L(M)$  for some Turing machine  $M$  such that:

- 1 If  $w$  is in  $L$ , then  $M$  accepts (and therefore halts)
- 2 If  $w$  is not in  $L$ , then  $M$  eventually halts (in a non-final state)

A Turing machine of this type corresponds to our notion of “algorithm”, a well-defined sequence of steps that always terminates and produces an answer. When we think of the language  $L$  as a “problem”,  $L$  is called *decidable* if it is recursive, otherwise called *undecidable*.

# Overview



$L_d$ : A language that is not recursively enumerable

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

Preliminary steps:

- 1 Enumerating binary strings
- 2 Representing Turing machines in binary strings

# Enumerating Binary Strings

- A binary string can be represented by a unique integer  $i$ :

The integer for binary string  $w$  is the integer value of  $1w$ .

- $w_i$ : the  $i$ th binary string

$$w_1 = \epsilon$$

$$w_2 = 0$$

$$w_3 = 1$$

$$w_4 = 00$$

$$w_5 = 01$$

$$w_6 = 10$$

$$w_7 = 11$$

$$w_8 = 000$$

$$w_9 = 001$$

$\vdots$



## Representing Turing Machines as Binary Strings

$$M = (Q, \{0, 1\}, \Gamma, \delta, q_1, B, F)$$

- 1  $Q = \{q_1, q_2, \dots, q_r\}$
- 2  $\Gamma = \{X_1, X_2, X_3, \dots, X_s\}$
- 3 Directions :  $\{D_1, D_2\}$

Encoding for the transition function  $\delta(q_i, X_j) = (q_k, X_l, D_m)$ :

$$0^i 10^j 10^k 10^l 10^m$$

Encoding for the Turing machine  $M$ :

$$C_1 11 C_2 11 \dots C_{n-1} 11 C_n$$

( $C_i$ : encoding for the  $i$ th transition rule of  $M$ ).

## Example

$$M = (\{q_1, q_2, q_3\}, \{0, 1\}, \{0, 1, B\}, \delta, q_1, B, \{q_2\})$$

$$\delta(q_1, 1) = (q_3, 0, R), \quad 0100100010100$$

$$\delta(q_3, 0) = (q_1, 1, R), \quad 0001010100100$$

$$\delta(q_3, 1) = (q_2, 0, R), \quad 00010010010100$$

$$\delta(q_3, B) = (q_3, 1, L), \quad 0001000100010010$$

Encoding in binary string:

01001000101001100010101001001100010010010100110001000100010010

## Turing machines can be ordered

$M_i$ : The  $i$ th Turing machine

### Definition

We define  $M_i$  to be the Turing machine whose binary representation is  $w_i$ .

## Turing machines can be ordered

$M_i$ : The  $i$ th Turing machine

### Definition

We define  $M_i$  to be the Turing machine whose binary representation is  $w_i$ .

When  $M_i$  is not a valid Turing machine, define  $M_i$  to be a Turing machine with one state and no transitions, e.g.,  $M_1$ .

# The Diagonalization Language

## Definition

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

## Theorem

*$L_d$  is not a recursively enumerable language.*

## Proof Sketch

Consider the table  $T$ :

		$j$				
		1	2	3	4	...
$i$	1	0	1	1	0	...
	2	1	1	0	0	...
	3	0	0	1	1	...
	4	0	1	0	1	...
	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	

- $T[i, j]$  is 1 iff TM  $M_i$  accepts input string  $w_j$ . The  $i$ th row is the characteristic vector for the language  $L(M_i)$ .
- To construct  $L_d$ , complement the diagonal.
- No Turing machine exists whose language is the diagonal language  $L_d$ , because the characteristic vector for  $L_d$  (i.e., the complemented diagonal) is different from the characteristic vectors of all Turing machines.
- The language of  $M_1$  is not  $L_d$  because  $L(M_1)$  differs from  $L_d$  for  $w_1$ , the language of  $M_2$  differs from  $L_d$  for  $w_2$ , and so on.

## $L_u$ : A language that is RE but not recursive

The following language is recursively enumerable but not recursive:

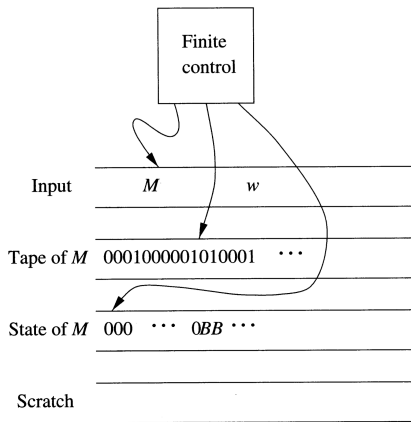
$$L_u = \{(M, w) \mid w \in L(M)\}$$

where  $M$  denotes Turing machines in binary strings.  $L_u$  is the set of strings representing a TM and an input accepted by the TM.

We can show that there is a Turing machine  $U$ , called *universal Turing machine*, such that  $L_u = L(U)$ . Given a Turing machine  $M$  and its input  $w$ ,  $U$  executes  $M$  on  $w$  and checks whether the machine  $M$  accepts the string  $w$  or not.

# Universal Turing Machine

A multitape Turing machine:





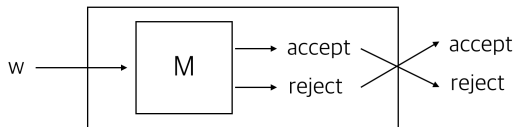
# Universal Turing Machine

- In the first tape, the binary representation of  $M$  (i.e. transitions of  $M$ ) and the input string  $w$  are stored.
- The second tape will be used to hold the simulated tape of  $M$ , where the tape symbols are represented by binary strings. That is, tape symbol  $X_i$  of  $M$  will be represented by  $0^i$ , and tape symbols are separated by single 1's.
- The third tape of  $U$  holds the state of  $M$ , with state  $q_i$  represented by  $i$  0's.
- To simulate a move of  $M$ ,  $U$  searches on its first tape for a transition  $0^i 1 0^j 1 0^k 1 0^l 1 0^m$ , such that  $0^i$  is the state on tape 3, and  $0^j$  is the tape symbol of  $M$  that begins at the position on tape 2 scanned by  $U$ . This transition is the one  $M$  would next make.  $U$  should:
  - 1 Change the contents of tape 3 to  $0^k$ .
  - 2 Replace  $0^j$  on tape 2 by  $0^l$ .
  - 3 Move the head on tape 2 to the position of the next 1 to the left or right, respectively, depending on whether  $m = 1$  (move left) or  $m = 2$  (move right).
- $U$  accepts the coded pair  $(M, w)$  if and only if  $M$  accepts  $w$ .

# Properties of complements (1)

## Lemma

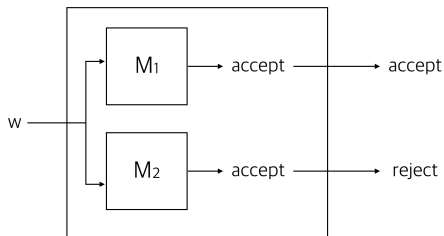
*If  $L$  is a recursive language, then so is  $\bar{L}$ .*



## Properties of Complements (2)

### Lemma

*If both a language  $L$  and its complement are RE, then  $L$  is recursive.*



## $L_u$ is not recursive

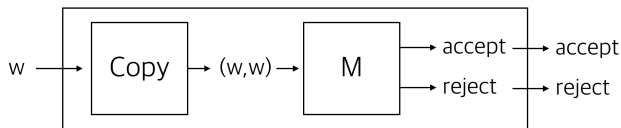
### Theorem

$L_u$  is RE but not recursive.

- Suppose  $L_u$  were recursive.
- Then by the property of complements,  $\bar{L}_u$  is also recursive.
- However, if we have a TM  $M$  to accept  $\bar{L}_u$ , then we can construct a TM to accept  $L_d$  (explained next).
- We already know that  $L_d$  is not RE, contradiction.

## Construction of TM to accept $L_d$ from TM to accept $\bar{L}_u$

Suppose  $L(M) = \bar{L}_u$ . We construct  $M'$  s.t.  $L(M') = L_d$  as follows:



# Summary

We have

- ① defined the class of recursively enumerable languages,
- ② defined the class of recursive languages,
- ③ defined a non-recursively enumerable language  $L_d$  and prove it, and
- ④ defined a non-recursive language  $L_u$  and prove it.