CS 3313
Foundations of Computing:

Undecidable Problems and Rice's Theorem

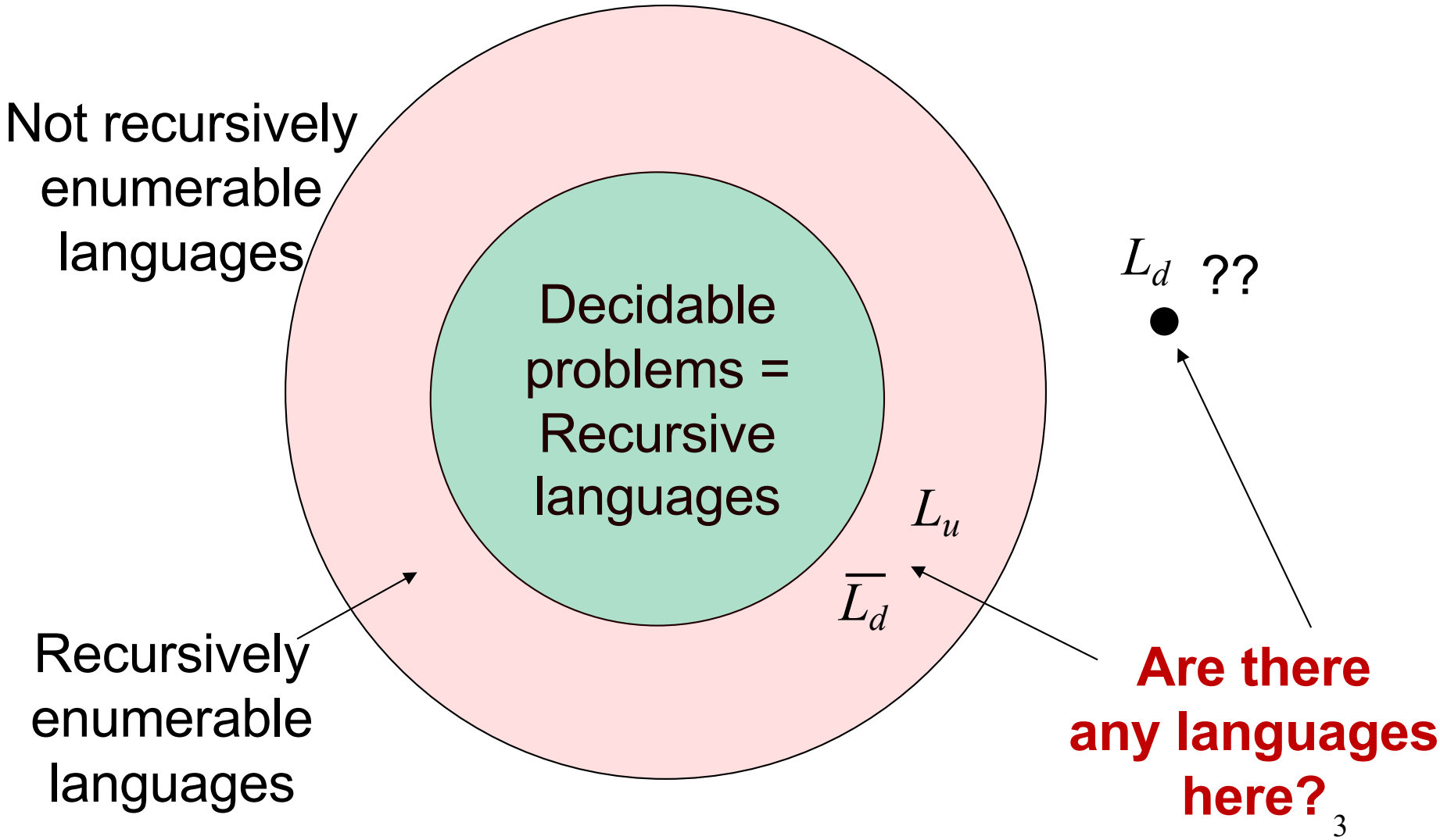
http://gw-cs3313-2021.github.io

Decidable Problems

- A problem is *decidable* if there is an algorithm to answer it.
  - Recall: An "algorithm," formally, is a TM that halts on all inputs, accepted or not.
  - Put another way, "decidable problem" = "recursive language."
- Otherwise, the problem is *undecidable*.

- Language is recursive if it is accepted by a TM that halts on all inputs.
- Language is recursively enumerable (r.e.) if it is accepted by a TM
  - TM halts and accepts if the string is in the language
  - However, TM may not halt if the string is not in the language

Undecidable Problems



A key proof technique: Reducability

- Reducibility of a problem A to problem B
- Given two problems A and B,

  problem A is reducible to problem B if an algorithm for solving B can be used to solve problem A

  – Therefore, solving A cannot be harder than solving B

  - *If A is undecidable and A is reducible to B, then B is undecidable*
- Idea: If you had a black box that can solve instances of B, can you solve instances of A using calls to this Black box.
  - The black box is the assumed Algorithm for B.

Today….

- Review our undecidability results/proofs
  - One more example
- The Post correspondence problem
- Rice's Theorem: Undecidability properties of r.e. languages

Our current "collection" of undecidable languages

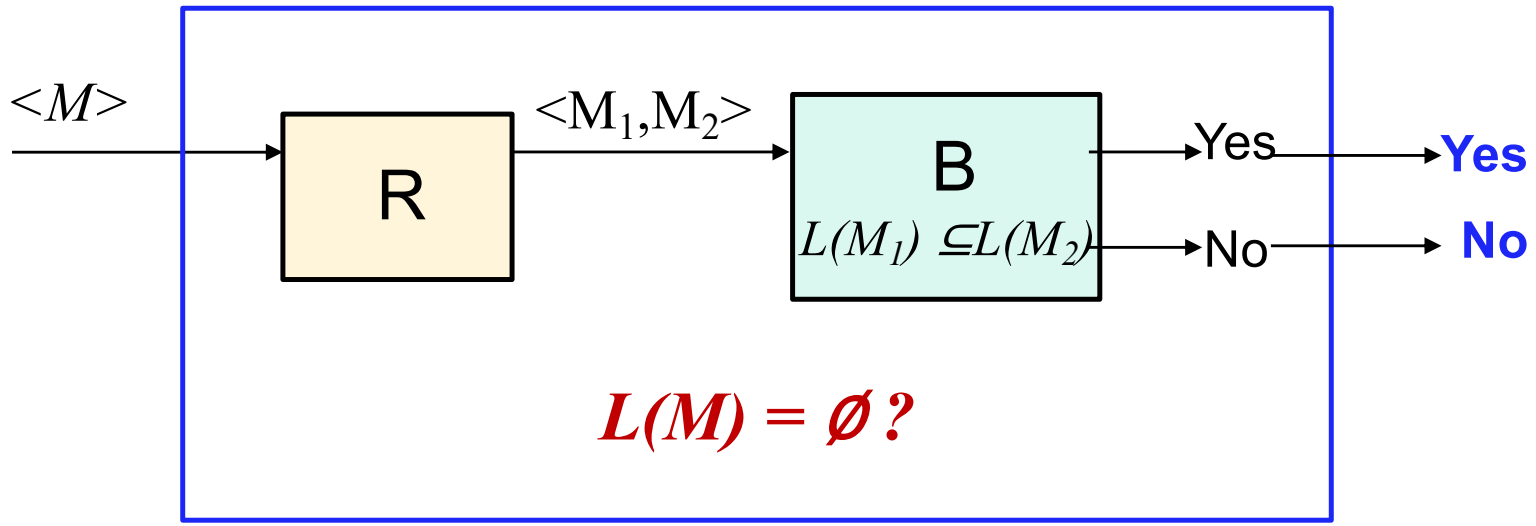
1. We proved that L_d *is not decidable (it is not even r.e.)*
   - $L_d = \{w \mid w = w_i$ and M_i does not accept $w_i\}$.
2. If L_d is not recursive then its complement $\overline{L_d}$ is not recursive, i.e, it is undecidable
   - $\overline{L_d} = \{w \mid w = w_i$ and M_i accepts $w_i\}$.

1. $L_u = \{ <M,w> \mid M$ accepts $w \}$ ....*Halting Problem*
   - We reduced $\overline{L_d}$ to L_u
2. Does M halt on all inputs is undecidable
   - We reduced L_u to this problem.
3. $L_e = \{ <M> \mid L(M) = \emptyset\}$ – Given any TM M, does M accept empty set. *In lab today*

Example 2/Exercise: { <M1,M2> | $L(M_1) \subseteq L(M_2)$}

- Given any two Turing machines M_1, M_2 does is the language accepted by M_1 a subset of language accepted by M_2 ?
- Hint 1: Reduce Emptiness problem to TM Subset problem.

- Hint 2: Recall set properties (subset)

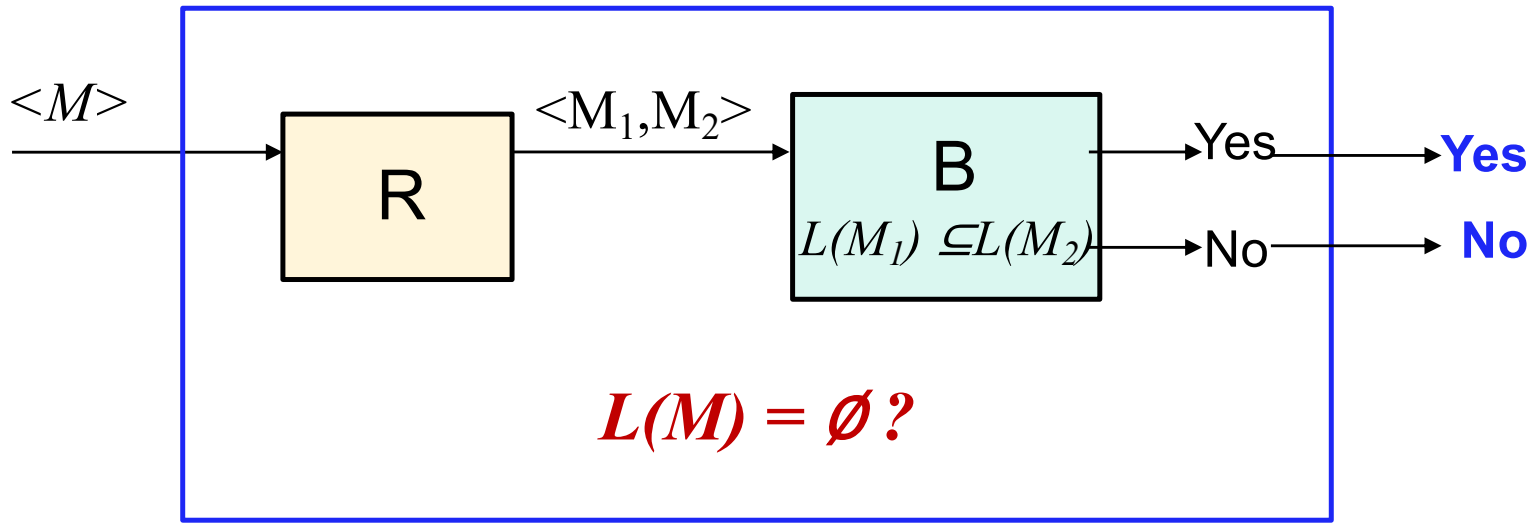


Example 2/Exercise: { <M1,M2> | $L(M_1) \subseteq L(M_2)$}

- Algorithm R:

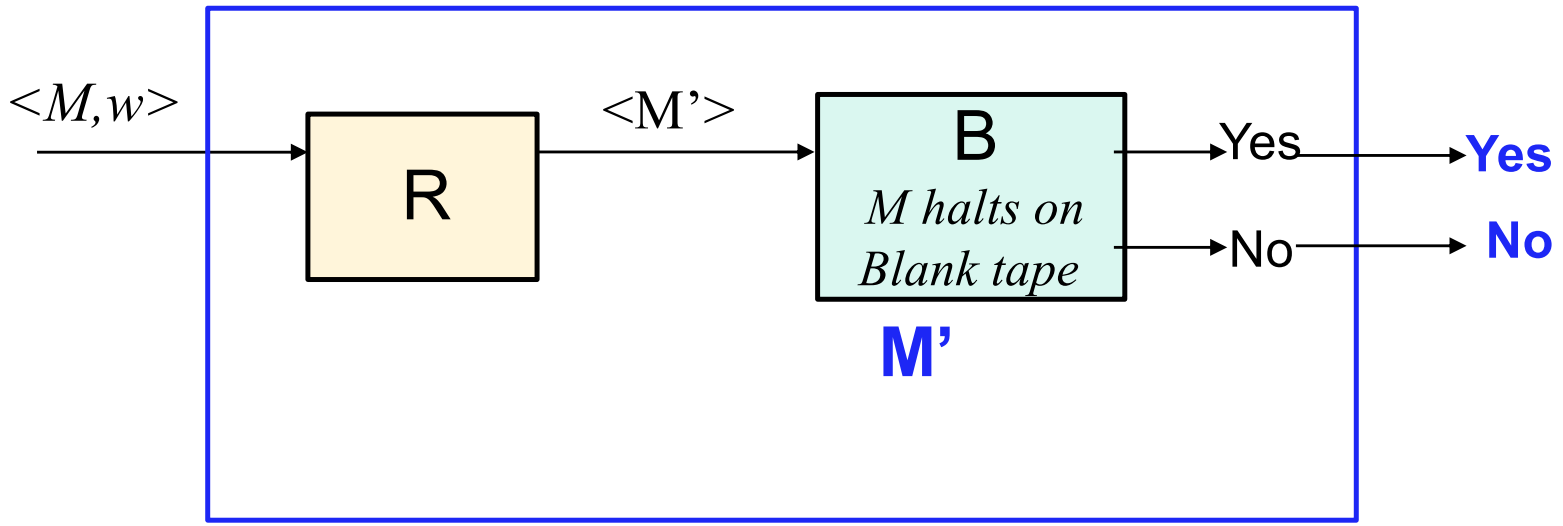
1. Generate TM M_2 such that $L(M_2)= \emptyset$
2. Copy M from input and set $M_1= M$

- Send (M_1,M_2) to Algorithm B:
  - If B answers yes then $L(M_1) \subseteq L(M_2)$. Since $L(M_2)=\emptyset$, we have $L(M_1)= \emptyset$
  - If B answers No then $L(M_1)$ is not empty



Example 3: Blank tape acceptance {<M>| M halts on blank tape}

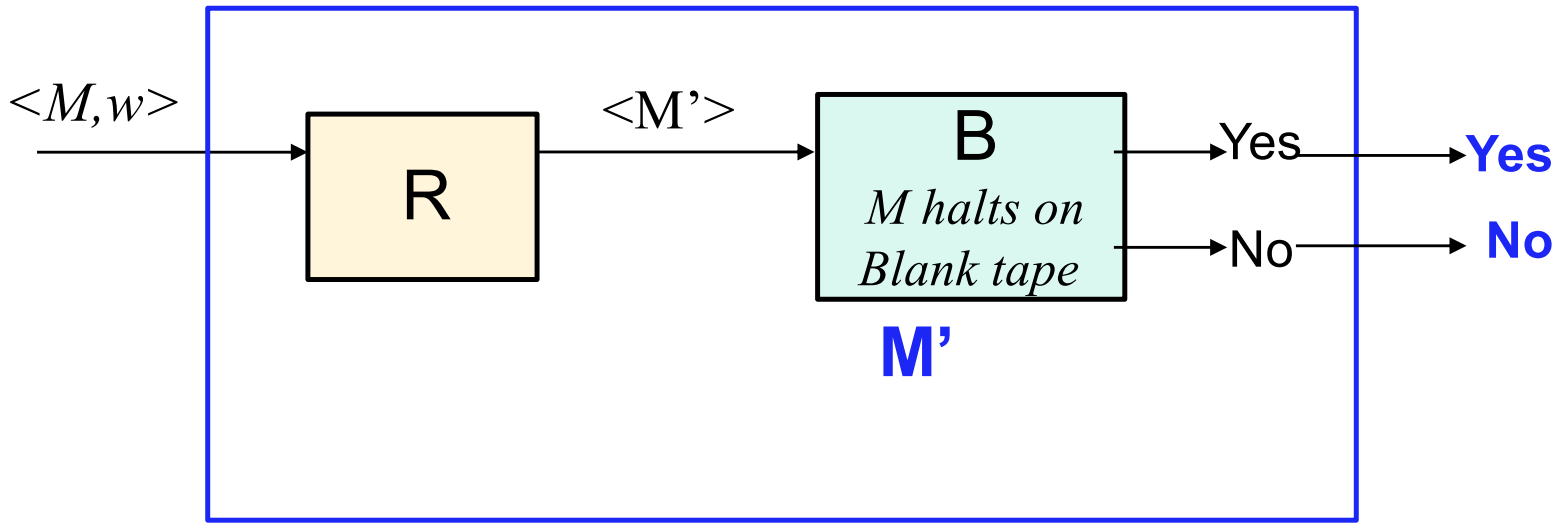
- Does M halt when started with the blank tape ?
- Can reduce this problem to the halting problem using a reducibility algorithm similar (identical?) to what we did for the Emptiness problem



Example 3: Blank tape acceptance {<M>| M halts on blank tape}

- Algorithm R: Generates modified TM *M' where*

1. M' first writes w to the tape.
2. Go to start state of *M*
3. *M'* accepts any input iff *M* accepts *w* – final state of *M'* is final state of *M*



Example 4: Does M print specific symbol *a* to the tape ?

See textbook for details

Examples: Undecidable Problems

- Does a program halt on all inputs ?
  - Can we build a compiler that checks this.
- Does a program enter a specific ‘checkpoint’– the state entry problem (does TM enter a specific state)?
- Can a particular line of code in a program ever be executed?
  - Version of question in Homework 8 !
- Post Correspondence Problem
  - Is a given context-free grammar ambiguous?
  - Do two given CFG’s generate the same language?
- Properties of r.e. sets -- Is the language accepted by a TM a regular language ?

Distraction intro reality….

- Seemingly a lot of useful questions about programs are undecidable
  - Does it halt on all inputs, does it enter a specific state/breakpoint, does a program print a specific string, etc.

- So how do you deal with software design and program correctness when you such decisions are undecidable ?
  - Note: there are some properties for which program verification tools exist

The Post Correspondence Problem (PCP)

- Given two sequences of n strings on some alphabet Σ, for instance

  $A = w_1, w_2, \ldots, w_n$ and $B = v_1, v_2, \ldots, v_n$

  there is a Post correspondence solution (PC solution) for the pair (A, B) if there is a nonempty sequence of integers

  $i, j, \ldots, k,$ such that $w_i w_j \ldots w_k = v_i v_j \ldots v_k$

- Example: assume A,B are

  $w_1 = 11,$ $w_2, = 10111,$ $w_3 = 10$

  $v_1 = 111$ $, v_2, = 10,$ $v_3 = 0$

  solution for this instance of (A, B) exists: sequence 2113

  $w_2 w_1 w_1 w_3 =$ 10111111110

  $v_2 v_1 v_1 v_3 =$ 10111111110

The Undecidability of the Post Correspondence Problem

- The Post correspondence problem is to devise an algorithm that determines, for any (A, B) pair, whether or not there exists a PC solution
- For example, there is no PC solution if A and B consist of

$w_1 = 00,\ w_2, = 001,\ w_3 = 1000$ and $v_1 = 0,\ v_2, = 11,\ v_3 = 011$

- **Theorem:** the Post correspondence problem (PCP) is undecidable
- result is crucial for showing the undecidability of various problems involving context-free languages

Undecidable Problems for Context-Free Languages

- The Post correspondence problem is a convenient tool to study some questions involving context-free languages
- The following questions, among others, can be shown to be undecidable
  - Given an arbitrary context-free grammar G, is G ambiguous?
  - Given arbitrary context-free grammars G_1 and G_2,
    is $L(G_1) \cap L(G_2) = \varnothing$?
  - Given arbitrary context-free grammars G_1 and G_2,
    is $L(G_1) = L(G_2)$?
  - Given arbitrary context-free grammars G_1 and G_2,
    is $L(G_1) \subseteq L(G_2)$?

Properties of Recursively Enumerable Languages

- A language is r.e. if it is accepted by some TM M
  - A language is regular if it is accepted by some DFA M
  - A language is context free if it is accepted by some PDA M (or generated by a CFG G)
- Consider languages that discuss properties of r.e. languages…

  i.e., the languages are sets of TM codes such that membership of $<M>$ in the language depends only on $L(M)$ and not on M itself.

Properties of a language

- Let $\mathbb{P}$ be a set of *r.e.* languages, each is a subset of {0,1}* (or any alphabet) – $\mathbb{P}$ is said to be a property of *r.e.* languages.
- a set L has property $\mathbb{P}$ if L is an element of $\mathbb{P}$
- In terms of properties of the language accepted by a turing machine, let $L_{\mathbb{P}} = \{ \langle M \rangle \mid L(M) \text{ is in } \mathbb{P} \}$

Trivial and Non-trivial Properties

- Non-trivial property: refers to a property satisfied by some but not all r.e. languages
- Trivial property: property satisfied by all or none (of r.e. languages)

- More formally:
- P is a *trivial property* if P is empty or P consists of all *r.e.* languages
- P is a *non-trivial property* otherwise.

Rice's Theorem

- Rice's Theorem: Any non-trivial property **P** of r.e. languages is undecidable.

- So how does one use this result……
  - Observe that this theorem is about r.e. languages -- languages which are accepted by a TM
    - We can give a TM to accept a language in this set of languages

Rice's Theorem: Examples

- Determining if *{<M> | L(M) is finite}* is undecidable.
- Proof – we want to prove by using Rice's theorem.
- All we have to show is that this is a non-trivial property
  - How ?

Rice's Theorem: Example 2

Alternate proof for Emptiness Problem

- *{<M> | L(M) is empty}* is undecidable.
- Proof – we want to prove by using Rice's theorem.
- To show that this is a non-trivial property
  - ????

In-Class (your very last in-class….)

- Prove that there is no algorithm that can decide if a language accepted by a TM is a regular language.
- { <M> | L(M) is regular }

- If this were decidable, then you would not need to use the pumping lemma to show a language is not regular !!