CS 3313
Foundations of Computing:

Undecidable Problems

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

Recall: Design of Universal Turing Machine (UTM)

- Every TM is encoded as a binary string
  - Decoding is unique
- UTM takes as input $<M,w>$ and simulates M on w.
  - Input is binary encoding of M followed by string w
  - UTM accepts and halts if and only if M accepts w (and halts)

Enumeration:

- since TM M encoded as a binary string we can talk of the i-th Turing machine M_i
  - Integer j represented in binary is the code of M_j: *denoted* $<M_j>$
- We can also refer to the *i-th* string w_i

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.

Reducibility

- *Why is it useful*: Find one undecidable problem A, and then to show B is undecidable we construct a solution for A if we assume B is solvable/decidable.
  - To show a problem B is undecidable: Start with one undecidable problem A, and reduce other problems to that.

- In context of time complexity: if we can reduce problem A to B in polynomial time and A is NP-complete then B is NP-complete.
  - To show a problem B is NP-complete, start with a NPC problem A (such as SAT) and show a polynomial time reduction to B

Our current "collection" of undecidable languages

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

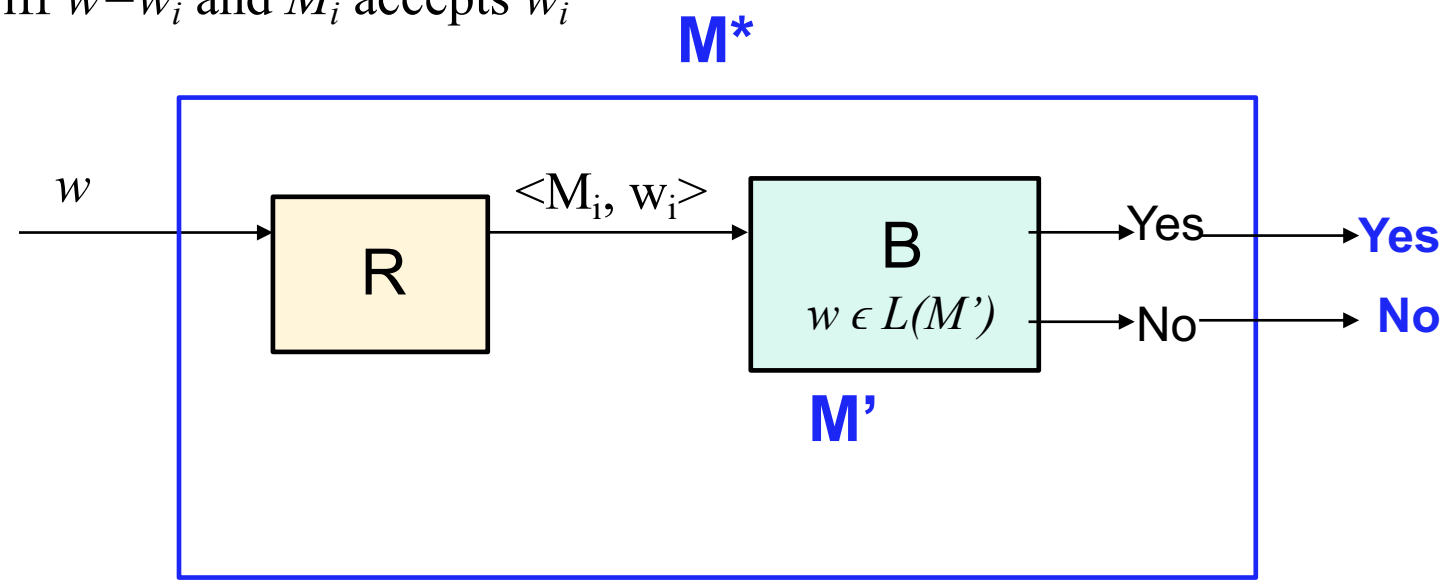
1. $L_u = \{ <M,w> \mid M \text{ accepts } w \}$ *....Halting Problem*
   - We reduced $\overline{L_d}$ to L_u (note the "direction")
2. Does M halt on all inputs is undecidable
   - We reduced L_u to this problem.

Recall Proof that L_u is not Recursive:

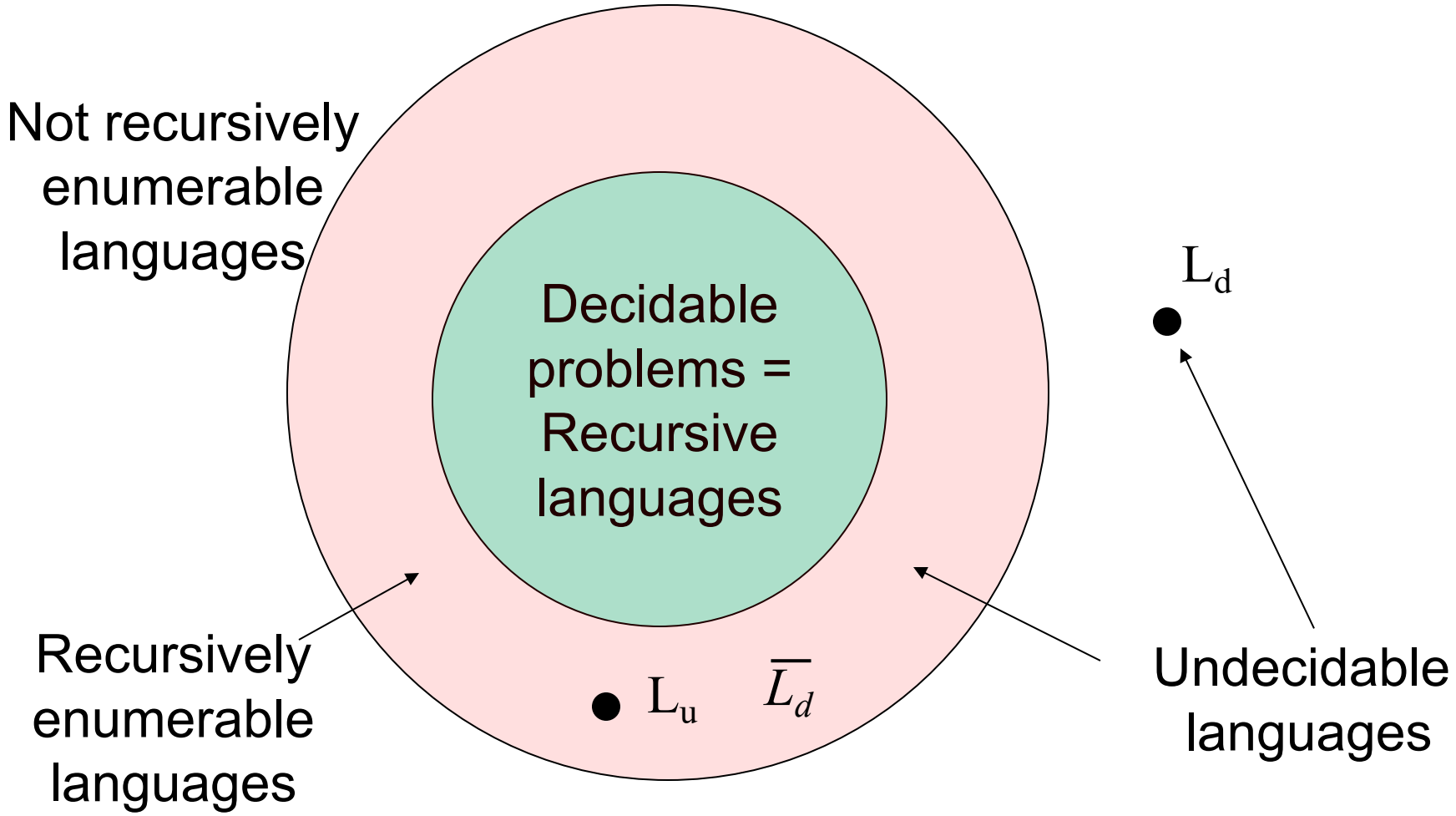
- algorithm R (the reduction): Input is w and output is $<M_i, w_i>$
  - Use the canonical ordering algorithm to find i, where $w = w_i$
  - Concatenate code for M_i and w_i to generate $<M_i, w_i>$
- *Send to hypothetical algorithm B for Halting Problem*
  - *B accepts if and only if w is in* $\overline{L_d}$

The proof was all about construction of R!

$w \in L(M^*)$ iff $w = w_i$ and M_i accepts w_i



Undecidable Problems

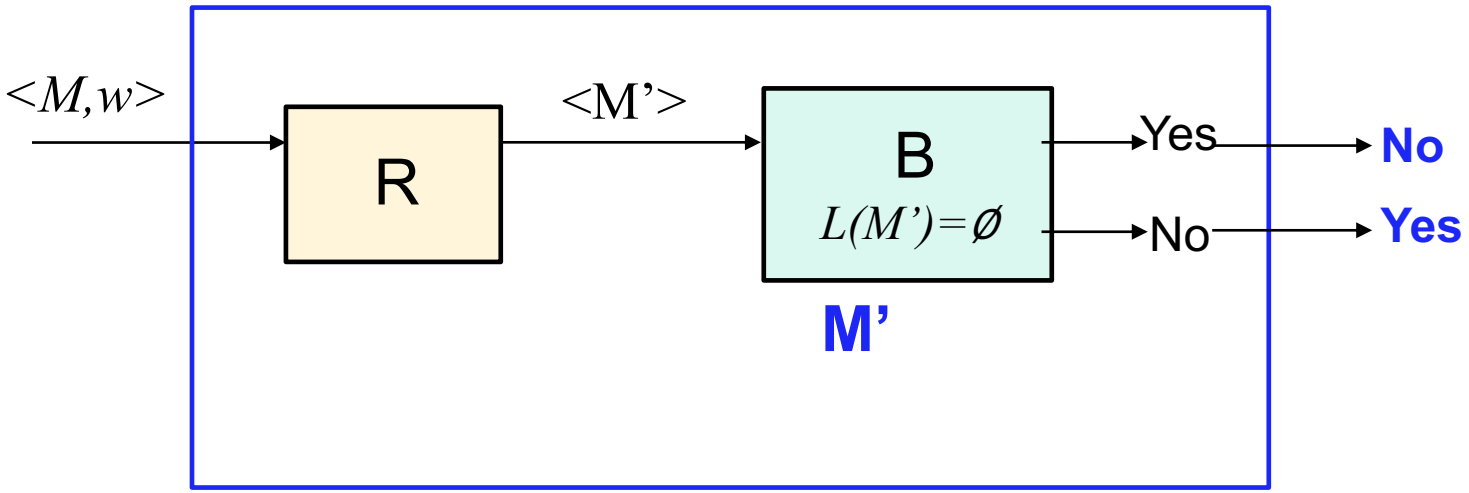


Today: Another example (or two or three?)

- Crucial step in the proof is the reduction “algorithm”
  - This process should be an “algorithm” – i.e., a TM that always halts
  - A very strict proof would require that one should construct a formal TM
    - We will live with constructing an algorithm, where we can reason that the steps in the algorithm can be carried out by a TM
    - To illustrate one complete example, we will show how a TM can be constructed in the reducibility step

Example 1: L = { <M> | M is a TM and L(M)=Ø }

- Question: Given a Turing machine M, does M accept any input ? (i.e., does M accept the empty set).
- Reducing L_u to Emptiness problem:
  - Assume Emptiness problem is decidable – implies there is an algo B that solves it
  - Construct algorithm R, such that testing for emptiness of M' using hypothetical algorithm B will give answer to "M accepts w".

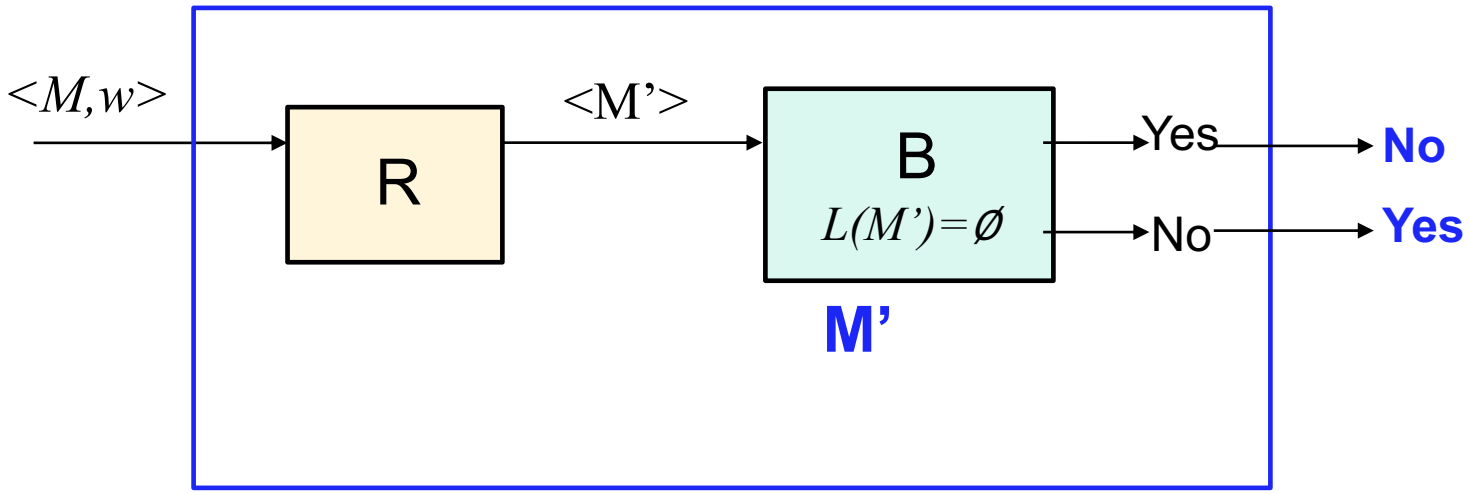


Example 1: L = { <M> | M is a TM and L(M)=Ø }

- Comment: What if instead of "M is a TM" we replace the problem with "M is a DFA"…..
- Question: Is this problem decidable ?.....
- Transition graph of DFA…Test if there is a path from start state to any of the final states.

Example 1: L = { <M> | M is a TM and L(M)=Ø }

- Reducibility of Halting problem to Emptiness problem
- Comment: Simply sending <*M*> to algorithm B can tell us if L(M) is empty. But if it is not empty then it does NOT mean w is accepted by M
- Therefore, have to send in a modified TM M' to Algo B, and emptiness of M' determines answer to "M accepts w"



Example 1: L = { <M> | M is a TM and L(M)=Ø }

- Key idea in constructing M': design M' such that M' accepts Ø iff M does not accept w, and M' accepts all strings ({0,1}*) iff M accepts w.
  - Design M'so that machine erases its input at the start, then writes w on the tape and starts M
- Modified TM *M'*:

1. For any input on the tape, replace *x* by *w*
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*

- *So what should reducibility algo R do:….generate M' !*

Example 1: L = { <M> | M is a TM and L(M)=Ø }

- Algorithm R: Input is <M,w> and output is M'

1. *Check length of w. Let length =n*
   - *w= $a_1a_2...a_n$ – note that this info is available from input <M,w>*
2. *Create n+3 states $q_1,q_2,...q_{n+3}$*
3. *Add (n+3) to indices of all states in M*
   - *Therefore start state of M now becomes q_{n+4} (original q_1 with n+3 added)*
4. *Start machine M', and replace any input x with string w*
5. *Accept if M accepts – final state of M' is final state of M*

- To illustrate one complete reducibility proof, let's examine how to construct a TM for Algorithm R

Example 1: L = { <M> | M is a TM and L(M)=Ø }

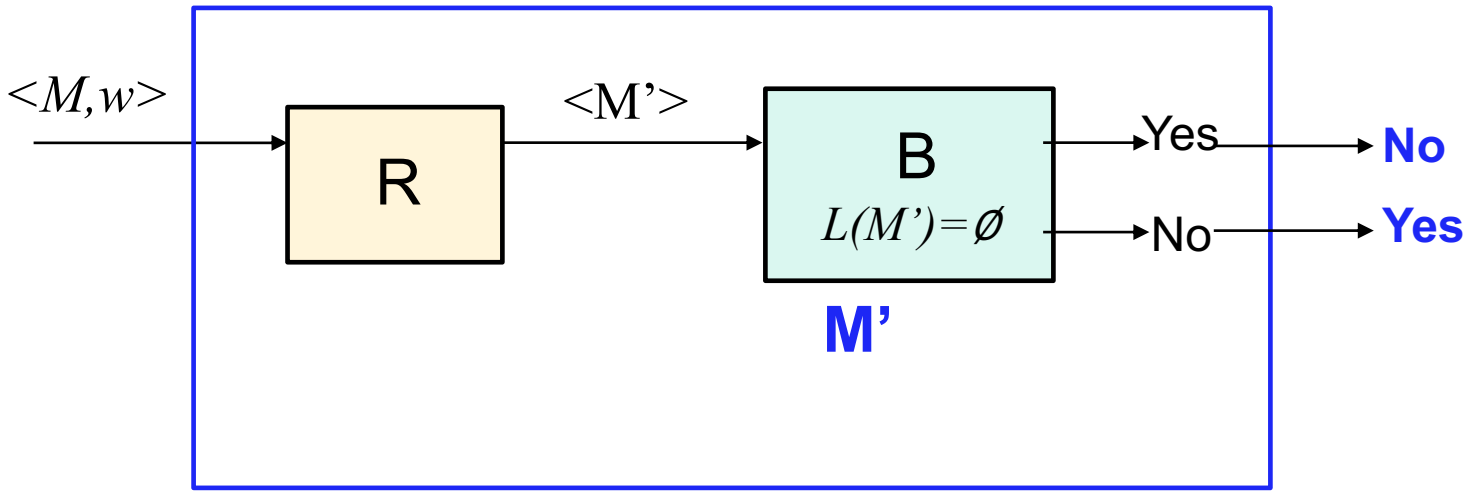
- TM to implement Algorithm R: Input is <M,w> and output is M'
- Details of step 4: $w = a_1 a_2 \ldots a_n$

1. $\delta(q_1,X) = (q_2, \$, R)$ for any X in Tape alphabet /* print marker $ at left end */
2. $\delta(q_2,X) = (q_2, a_1, R)$ for any X except B /* replace first symbol of tape with first symbol of w */
3. …
4. $\delta(q_i,X) = (q_{i+1}, a_{i-1}, R)$ for any X except B /*write $(i-1)$ symbol of w to tape in state q_i */
5. $\delta(q_{n+2},X) = (q_{n+2}, B, R)$ /* erase tape to right of w */
6. $\delta(q_{n+2},B) = (q_{n+3}, B, L)$ /* now move left to the $ marker */
7. $\delta(q_{n+3},B) = (q_{n+3}, B, L)$
8. $\delta(q_{n+3},\$) = (q_{n+4}, B, R)$ /* go to start state of M */
9. Add (n+3) to indices of all states in M and "update" transition function, i.e.,

- Ex: replace $\delta(q_j,X_1) = (q_k, X_2, L)$ *with* $\delta(q_{j+n+3},X_1) = (q_{k+n+3}, X_2, L)$

Example 1: L = { <M> | M is a TM and L(M)=Ø }

- If *M'* accepts any string *x*, then it erases tape, replaces with *w* and accepts *w* iff *M* accepts *w*.
- If *M'* does not accept any string iff *M* does not accept *w*
- Therefore *M accepts w iff L(M) is not empty*



Questions ?

Example 2/Exercise: $\{ \langle M1,M2 \rangle \mid L(M_1) \subseteq L(M_2) \}$

- Given any two Turing machines M_1, M_2 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)