CS 3313
Foundations of Computing:

Undecidable Problems

http://gw-cs3313.github.io

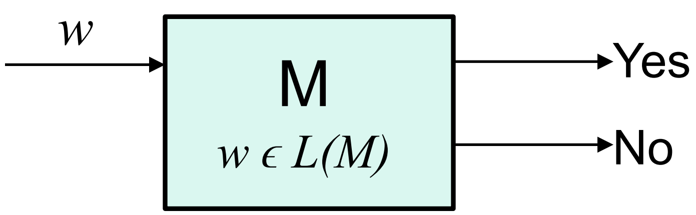
Decidable Problems

- 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*.

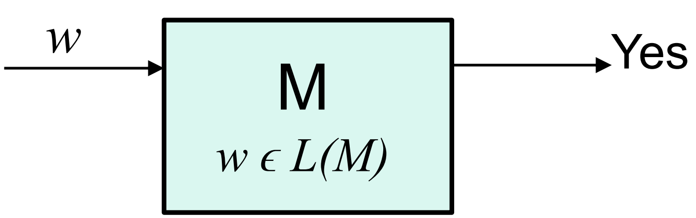
- Language is recursive if it is accepted by a TM that halts on all inputs.
- Language is recursively enumerable (r.e.) if it is accepted by a TM
  - TM halts and accepts if the string is in the language
  - However, TM may not halt if the string is not in the language

Recall Definitions

- Recursive Language: A language L is recursive if there is a Turing machine that accepts the language and *halts on all inputs*



- Recursively Enumerable Language: if there is a Turing machine that accepts the language by *halting when the input string is in the language*
  - The machine may or may not halt if the string is not in the language



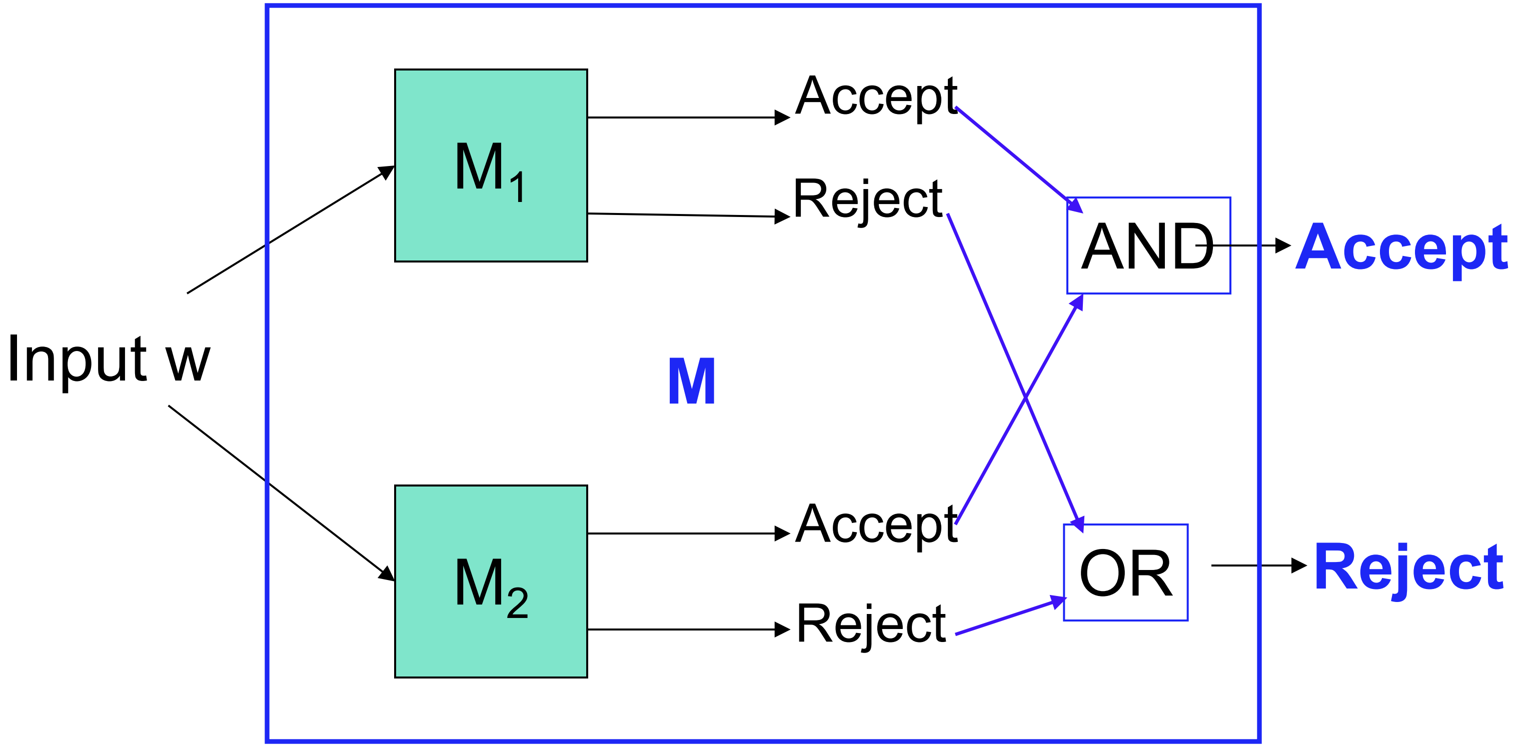
Review:
Closure Properties of Recursive and RE Languages

- Both are closed under union, concatenation, star, reversal, intersection, inverse homomorphism.
  - If L_1 and L_2 are recursive languages then so is their union, intersection, etc.
- Recursive closed under difference, complementation.
- RE closed under homomorphism.

- To prove closure properties, construct the algorithm (flowchart)

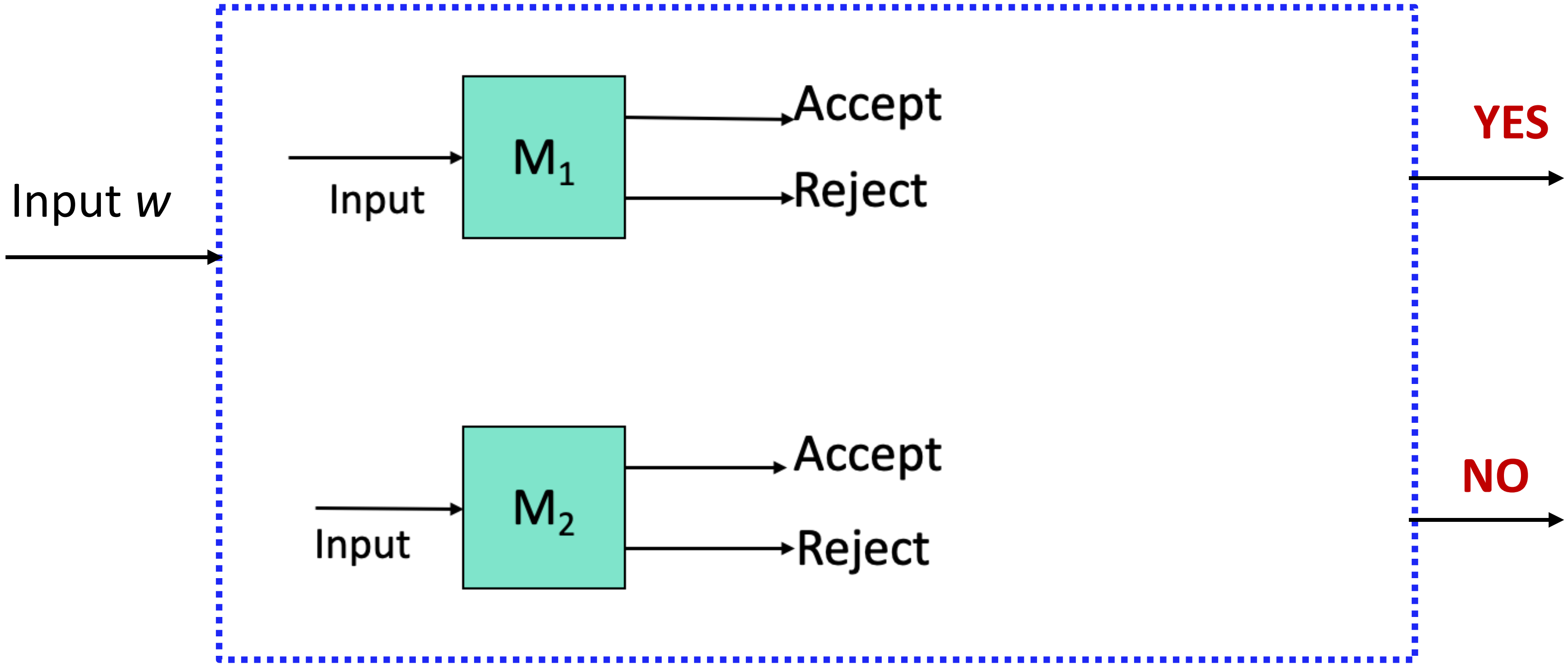
  Use ”output” (Yes or No answer) of Algorithm/TM for recursive language to construct algorithm for the new language (which is a set operation on recursive languages)

Example: Proof - Intersection of Recursive Languages is a Recursive Language



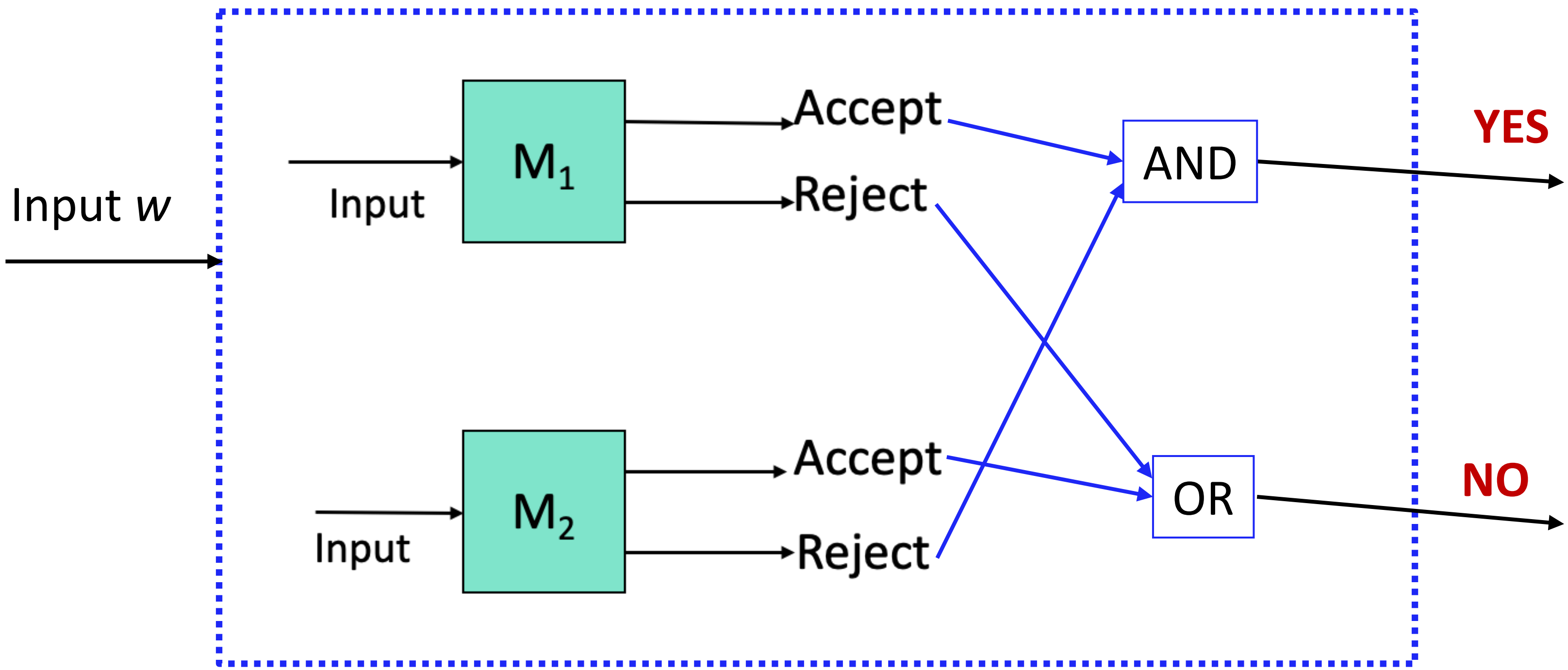
Exercise:
Recursive Languages are closed under Set Difference

$$L(M) = L(M_1) - L(M_2)$$



Exercise: Solution
Recursive Languages are closed under Set Difference

$$L(M) = L(M_1) - L(M_2)$$



Questions on Closure Properties of Recursive/R.E. Languages?

Review: Decidability…and Reducibility proof technique

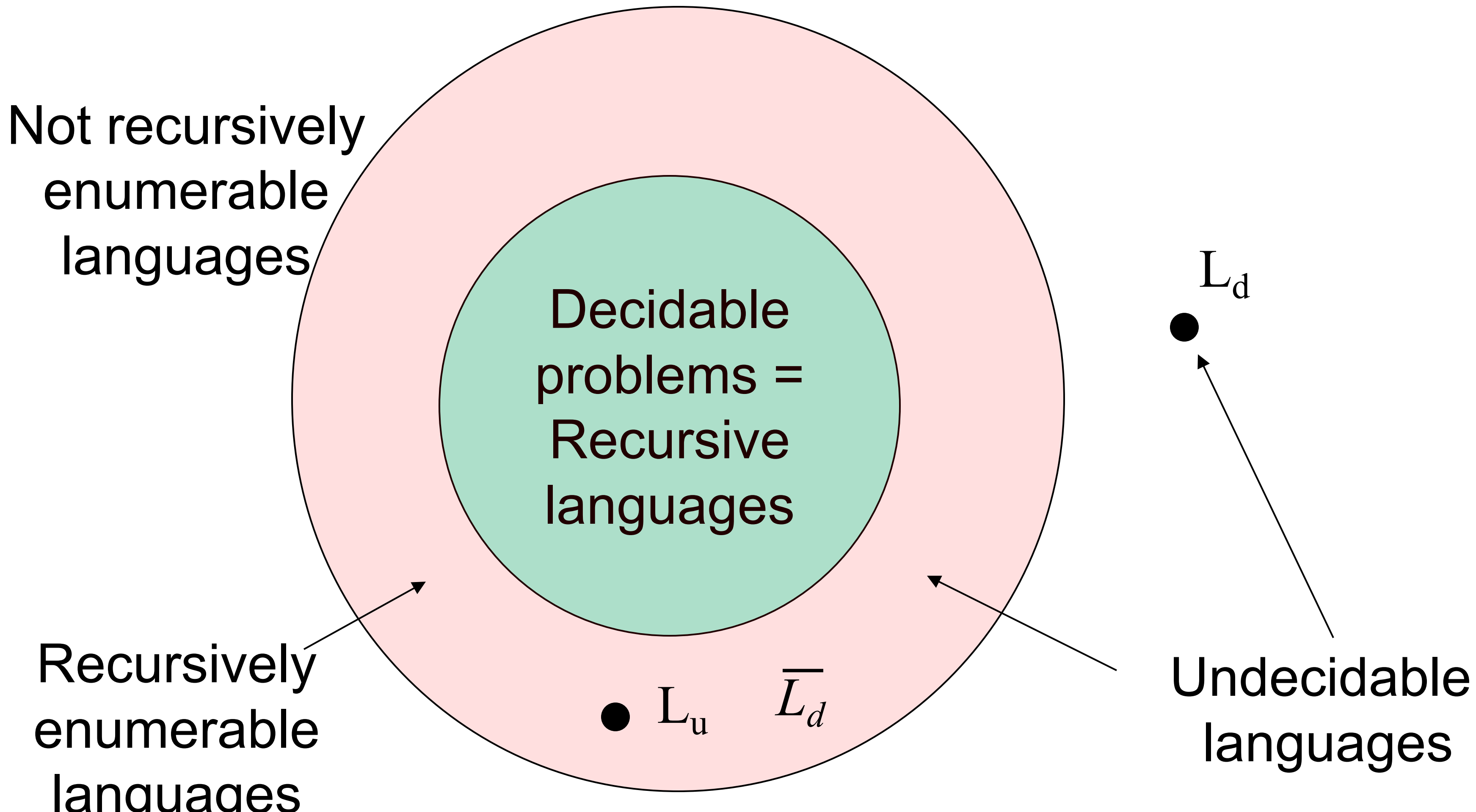
- Reducibility of a problem A to problem B
- Given two problems A and B,

  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.

Undecidable Problems



Our current “collection” of undecidable languages

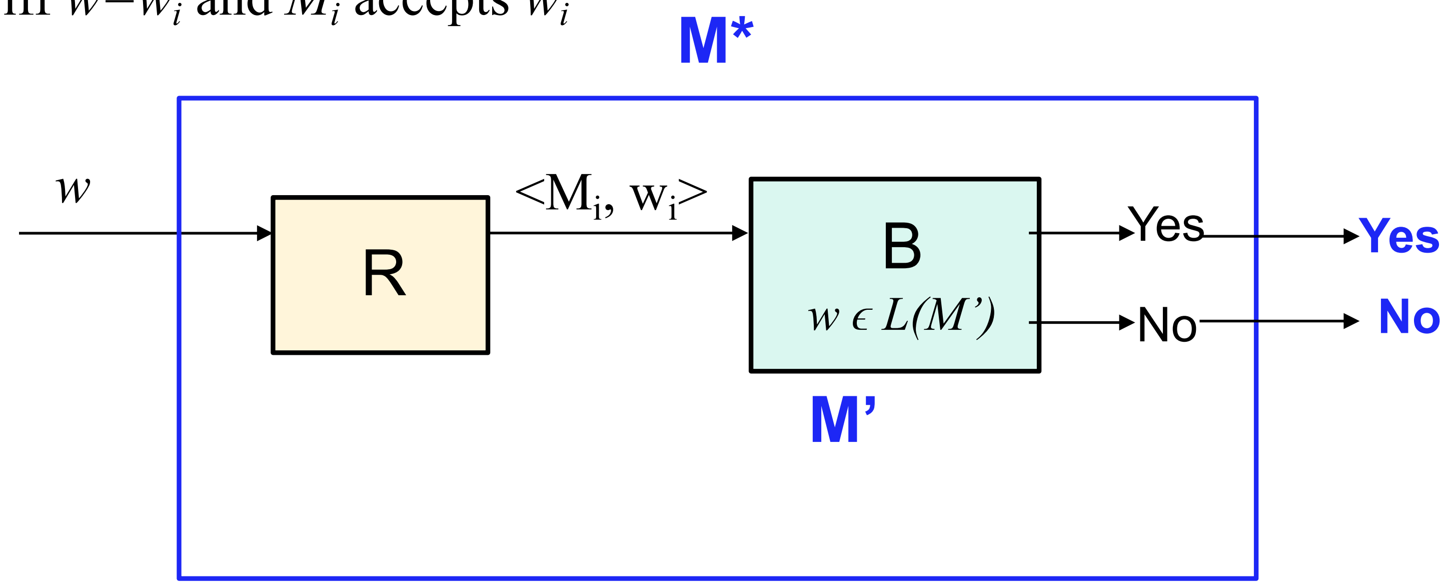
1. $L_d = \{w \mid w = w_i \text{ and } M_i \text{ does not accept } w_i\}$.
2. If $\overline{L_d} = \{w \mid w = w_i \text{ and } M_i \text{ accepts } w_i\}$.
3. $L_u = \{ <M,w> \mid M \text{ accepts } w \}$ ....*Halting Problem*
4. Does M halt on all inputs ?
5. Is L(M)= Ø
6. Is $L(M_1) = L(M_2)$
7. Does M accept blank tape ?
8. Does M reach a specific state ?
9. Does M reach a specific ID (snapshot) ?
10. Does M print a specific symbol ?

Recall Proof that L_u (halting problem) is undecidable

- $\overline{L_d} = \{\, w_i \mid M_i \text{ accepts (halts on) } w_i \,\}$
- Reducibility algorithm R ($\overline{L_d}$ reducible to B): 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 (binary representation of 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

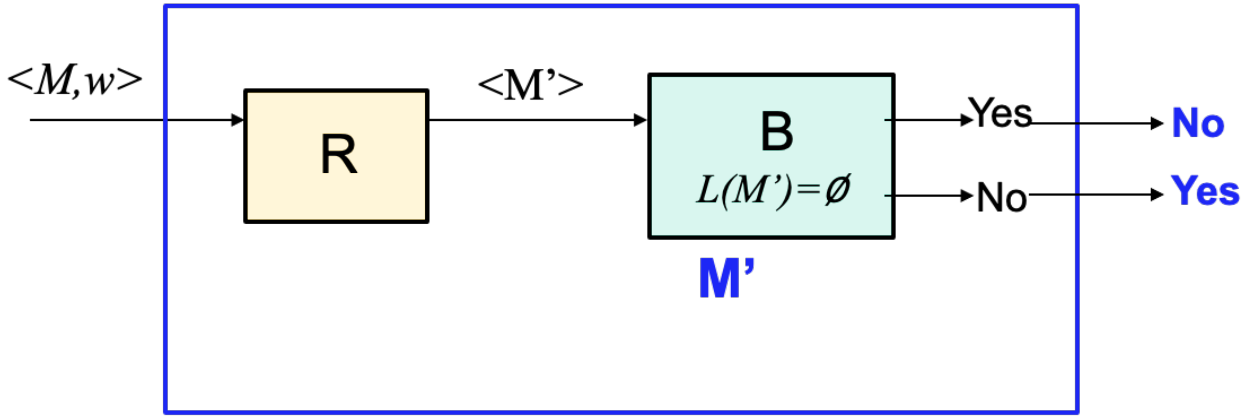


Today: Review another example (and one exercise)

- 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 reducibility algorithm (R) can be carried out by a TM
    - To illustrate one complete example, we outline/show how a TM can be constructed in the reducibility step

Example : 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 Halting problem 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*".
- 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: 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: L = { <M> | M is a TM and L(M)=Ø }

- Input to our hypothetical algorithm B is TM M'
  - And Algo B tells us if L(M') = Ø
- Input to Halting problem is <M,*w*> and the algorithm must determine if M accepts (halts on) *w*
- Modified TM *M'*:
- Key idea (in reducibility algorithm): machine will erase any input *x* on its tape and replace with w and then start M
  - Therefore, any string *x* is accepted by M' *iff* M accepts *w*
  - If any string is accepted then L(M') is not empty
  - If no string is accepted then L(M') is empty

Example: 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>

     $1110^i10^j10^k10^l10^m11.......111w$
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 is now q_{n+4} (original q_1 with n+3 added)
4. Start machine M', and replace any input x with string w
   - *Erase any input x on tape and write w on tape and then start M*
5. Accept if M accepts – final state of M' is final state of M

- Quick glimpse into how M' is constructed by Algorithm R

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

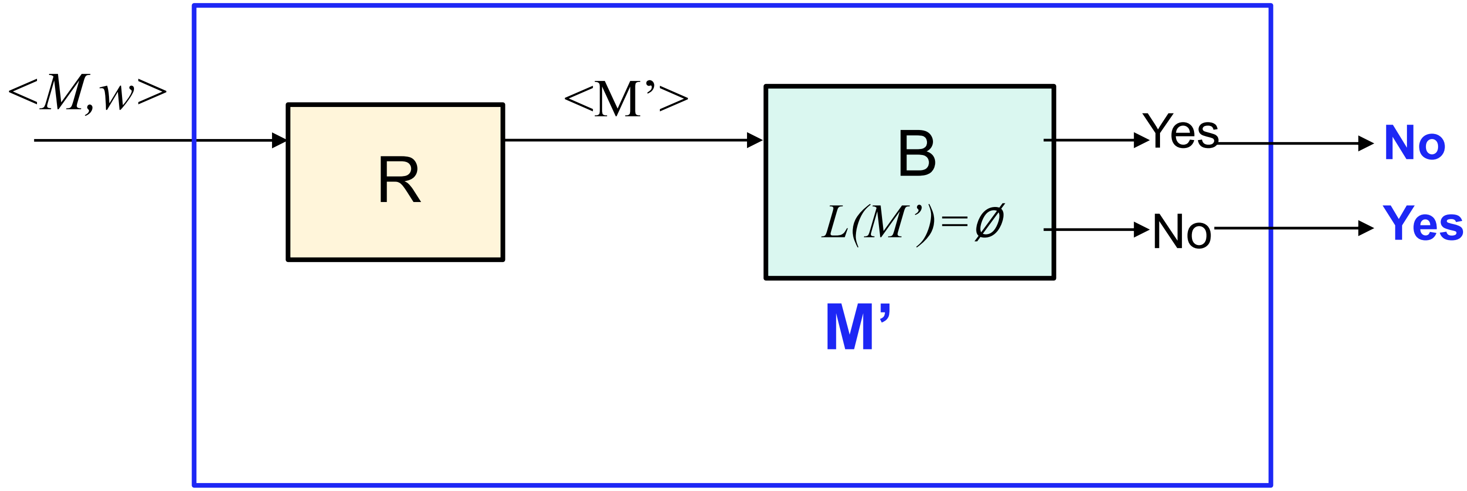
- TM to implement Algorithm R: Input is <M,w> and output is M'
- Details of step 2,3, 4: $w = a_1 a_2 \ldots a_n$

1. $\delta(q_1,X) = (q_2, \$, R)$ for any X in Tape alphabet /* print marker $ at left end */
2. $\delta(q_2,X) = (q_3, a_1, R)$ for any X except B /* replace first symbol of tape with first symbol of w */
3. $\delta(q_i,X) = (q_{i+1}, a_{i-1}, R)$ for any X except B /*write *(i-1)th* symbol of w to tape in state q_i */
4. $\delta(q_{n+1},X) = (q_{n+2}, a_n, R)$ /* write n-th symbol of w to tape */
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. $\delta(q_{n+3},B) = (q_{n+3}, B, L)$ *and* $\delta(q_{n+3},X) = (q_{n+3}, X, L)$
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: L = { <M> | M is a TM and L(M)=Ø }

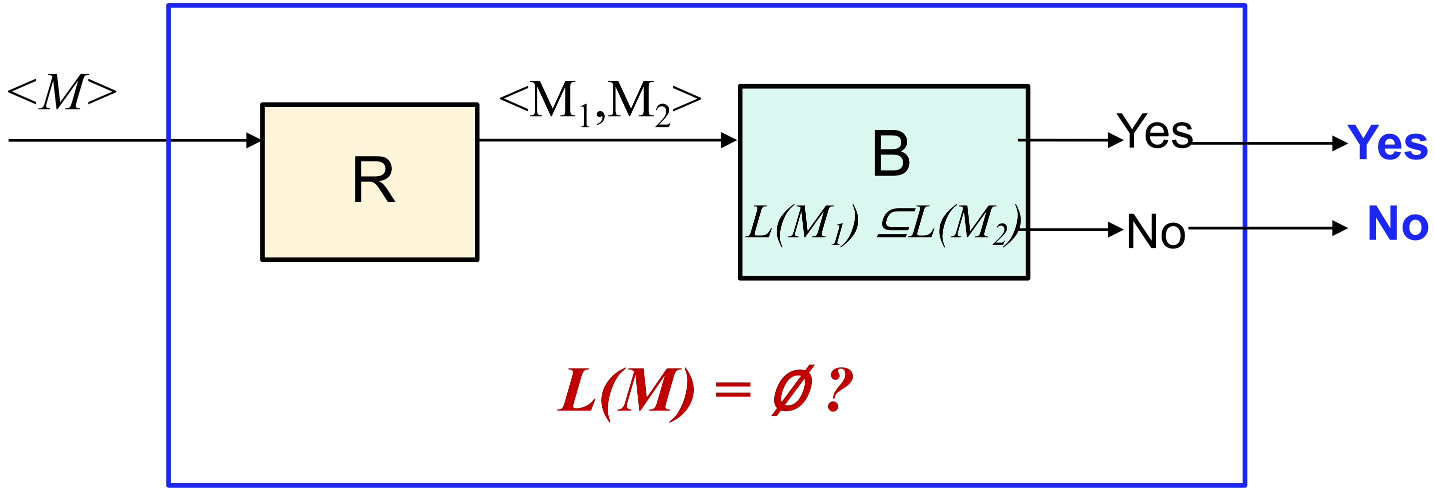
- If *M'* accepts any string *x*, then it erases tape, replaces with *w* and accepts *w* iff *M* accepts *w*.
- If *M'* does not accept any string iff *M* does not accept *w*
- Therefore *M accepts w iff L(M) is not empty*



Questions ?

Exercise: { <M1,M2> | $L(M_1) \subseteq L(M_2)$} is undecidable

- 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)
- Hint 3: Can you design a TM that accepts nothing (empty set)?

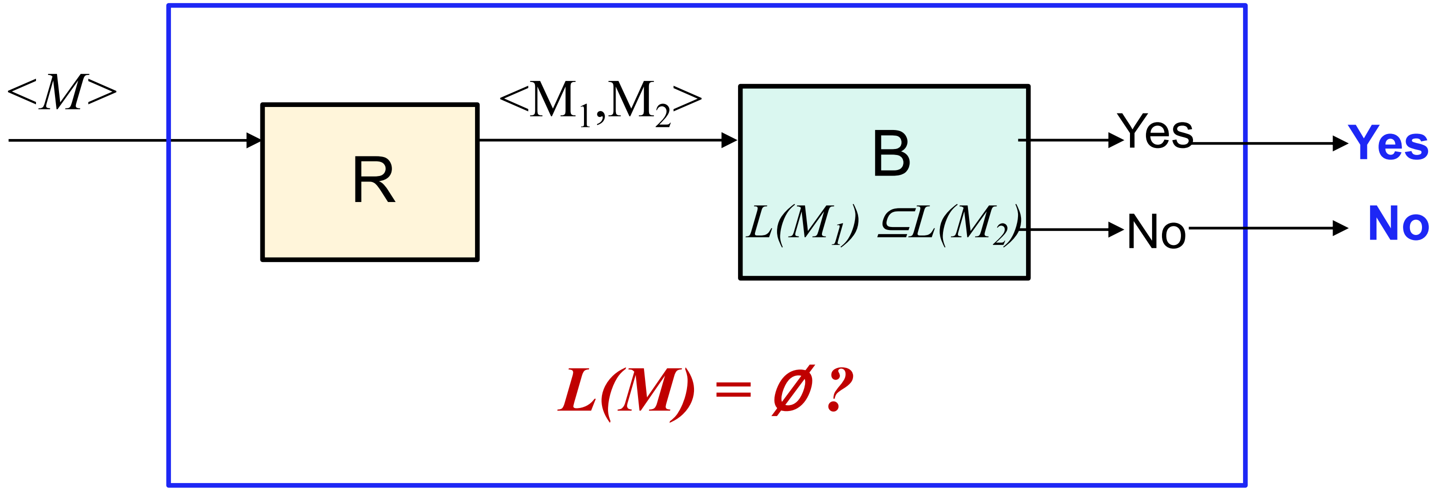


Solution: { <M1,M2> | $L(M_1) \subseteq L(M_2)$} is undecidable

- Algorithm R:

1. Generate TM M_2 such that $L(M_2) = \emptyset$ (no transition to final state)
   - $\delta(q_1, X) = (q_3, B, R)$ and $F = \{ q_2 \}$
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

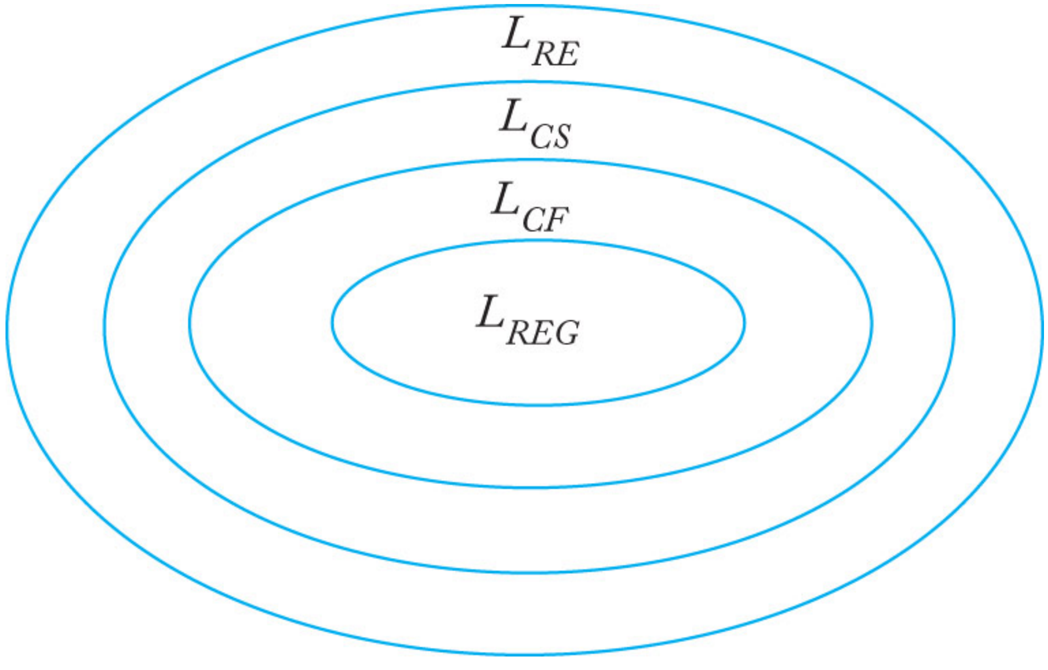


The Chomsky Hierarchy

- The linguist Noam Chomsky summarized the relationship between language families by classifying them into four language types, type 0 (regular lang) to type 3 -- the *Chomsky Hierarchy*
- In terms of automata:
- *DFA < PDA < TM*
- $L(DFA) \subset L(PDA) \subset L(TM)$
- *=> If DFA accepts L*

  then PDA accepts L
- *=> if PDA accepts L*

  then TM accepts L



Automata Models and The Chomsky Hierarchy

- Theorem: If G is a Regular grammar then L(G) is accepted by a DFA/Reg.Expression.
  - If L is accepted by a DFA then L =L(G) for some regular grammar G.
- Theorem: If G is a context free grammar, then L(G) is accepted by a PDA.
  - If L is accepted by a PDA then L = L(G) for some CFG G
- Theorem: If G is any unrestricted grammar then L(G) is accepted by a Turing machine.
  - All grammars are unrestricted grammars
  - Properties of unrestricted grammars = Properties of languages accepted by Turing machines!

- Theorem: If G is a context sensitive grammar then L(G) is accepted by a linear bounded automaton
  - A linear bounded automaton is a subclass of Turing machines

Questions on Chomsky Hierarchy ?

Review: Rice's Theorem

- Apply the theorem to prove if a property of r.e. languages is undecidable
- Important: This is not about the property of a Turing machine…Input is TM and the question is about the property of the language accepted by the TM

  question it asks is "is property of languages accepted by Turing machines decidable"

Properties of a language

- Let $\mathcal{P}$ be a set of *r.e.* languages, each is a subset of $\{0,1\}^*$ (or any alphabet) – $\mathcal{P}$ is said to be a property of *r.e.* languages.
- a set L has property $\mathcal{P}$ if L is an element of $\mathcal{P}$
  - Ex: If property $\mathcal{P}$ is "finiteness", then $\{a^i b^i \mid i < 10\}$ has property $\mathcal{P}$

    but $\{a^i\, b^i \mid i>0\}$ does not have property $\mathcal{P}$
  - Ex: If property $\mathcal{P}$ is "regular language", then $\{a^* b^*\}$ has property $\mathcal{P}$

    but $\{a^i\, b^i \mid i>0\}$ does not have property $\mathcal{P}$

- In terms of properties of the language accepted by a turing machine, let $L_{\mathcal{P}} = \{ <M> \mid L(M) \text{ is in } \mathcal{P} \}$

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:
- $\mathcal{P}$ is a *trivial property* if $\mathcal{P}$ is <u>empty</u> or $\mathcal{P}$ consists of <u>all</u> *r.e.* languages
- $\mathcal{P}$ is a *non-trivial property* otherwise.
  - Ex: Finiteness is a non-trivial property

Rice's Theorem

- Rice's Theorem: Any non-trivial property $\mathcal{P}$ 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: Example

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
  1. provide Turing machine that accepts a language with this property
     - Provide TM that accepts empty set
  2. Provide a TM that accepts a language without this property
     - Provide TM that accepts some string (non-empty language)

Exercise/Review: Testing Regularity of languages is undecidable

- 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 !!
- Why is such a question useful?
  - Let's say you have designed a program to solve a problem, and now want to know if that program can be implemented (in hardware) on a finite state machine.

Testing regularity of languages in Undecidable: Recall some examples we covered in the course

- Claim: Regularity of languages accepted by a TM is a non-trivial property
- $L_1 = (0+1)^*\ 00\ (0+1)^*$ accepts strings contain two consecutive 0's
  - L_1 is regular since we have a regular expression for the language
- $L_2 = \{ a^n b^n \mid n > 0 \}$ equal number of a's and b's
  - Proved it is not regular
  - Proved it is a CFL – designed PDA that accepts the language
  - Since anything accepted by a PDA is accepted by a TM, there is a TM that accepts $\{ a^n b^n \mid n > 0 \}$
- Therefore, this is a non-trivial property and by Rice's theorem it is undecidable !

Questions ?