CS 3313
Foundations of Computing:

Closure Properties of RE & Recursive Languages

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

Problems

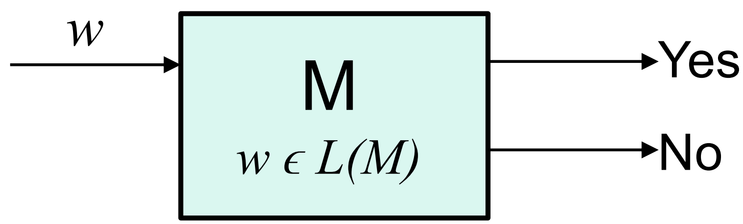
- Informally, a (decision) "problem" is a yes/no question about an infinite set of possible *instances*.
- Example 1: "Does graph G have a *Hamilton cycle* (cycle that touches each node exactly once)?
  - Each undirected graph is an instance of the "Hamilton-cycle problem."
- Example 2: "Is graph G k-colorable ?
  - Each undirected graph, and value k, is an instance of the "graph coloring problem."

Problems – (2)

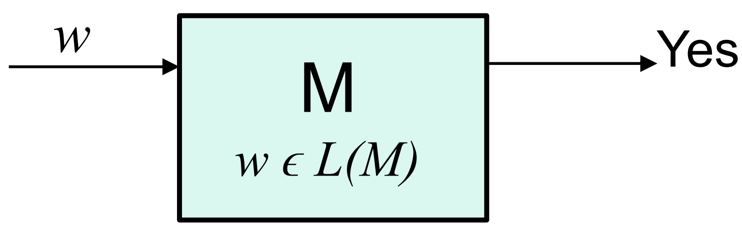
- Formally, a problem is a language.
- Each string encodes some instance.
- The string is in the language if and only if the answer to this instance of the problem is “yes.”

Recall Definitions

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



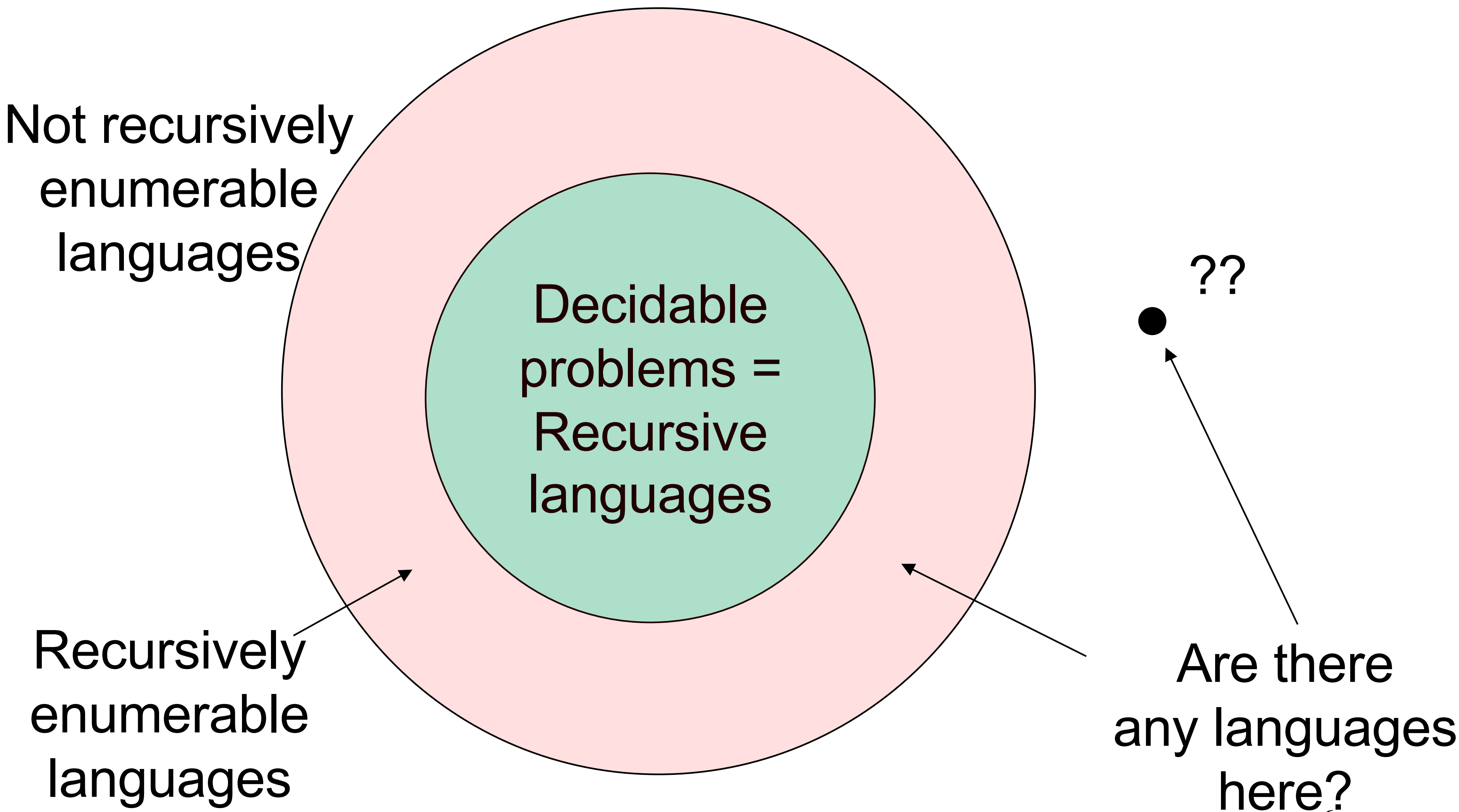
- Recursively Enumerable Language: if there is a Turing machine that accepts the language by halting when the input string is in the language
  - The machine may or may not halt if the string is not in the language



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*.

Bullseye Picture

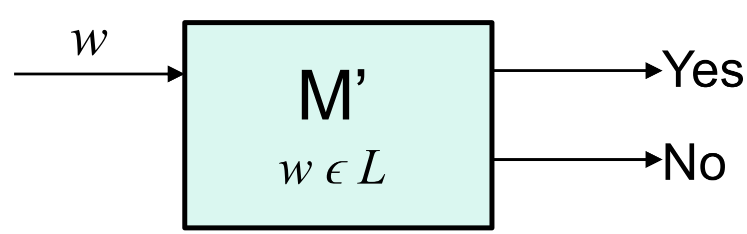


Closure Properties of Recursive and RE Languages

- Next topic is Decidability
  - Review Lab notes on Math review – diagonalization etc.
- First let's look at closure properties of these classes of languages
- <u>Both closed</u> under union, concatenation, star, reversal, intersection, inverse homomorphism.
- Recursive closed under difference, complementation.
- RE closed under homomorphism.

Proving Closure Properties…methodology

- *Observe: To prove the closure properties we have to construct a Turing machine, i.e., an algorithm (!!!), to accept the language*
  - *Construction shown using a flowchart & combining other ”algorithms”*
    - *Getting more and more like programming!*
- To prove a language L (constructed from other recursive languages) is recursive, provide an algorithm described by a ‘flowchart’ below
  - To show it is RE, the machine halts only if w is in the language

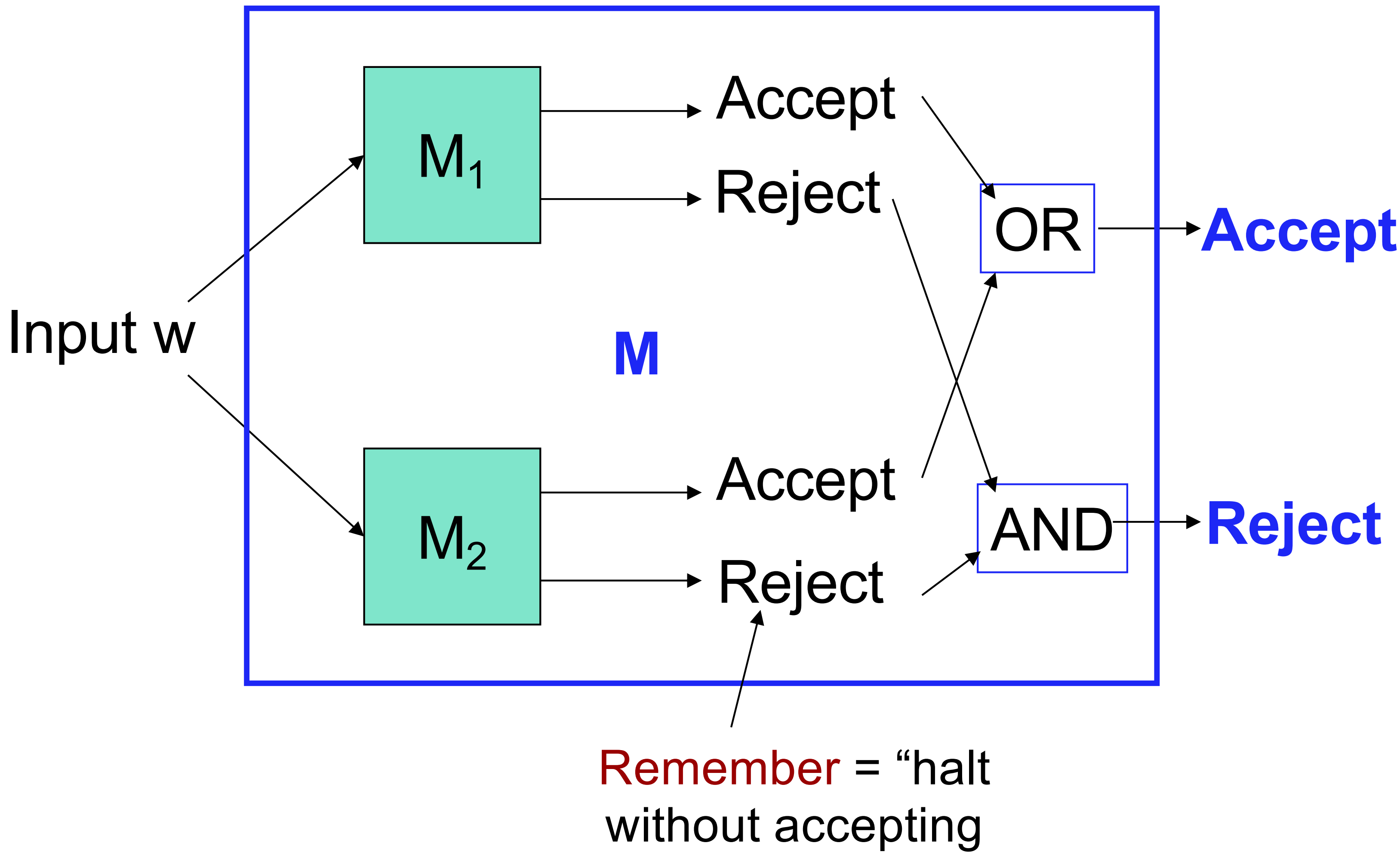


Closure under Union

- Let $L_1 = L(M_1)$ and $L_2 = L(M_2)$.
- Assume M_1 and M_2 are single-semi-infinite-tape TM's.
- Construct 2-tape TM M to copy its input onto the second tape and simulate the two TM's M_1 and M_2 each on one of the two tapes, "in parallel."
- Recursive languages: If M_1 and M_2 are both algorithms, then M will always halt in both simulations.
- RE languages: accept if either accepts, but you may find both TM's run forever without halting or accepting.

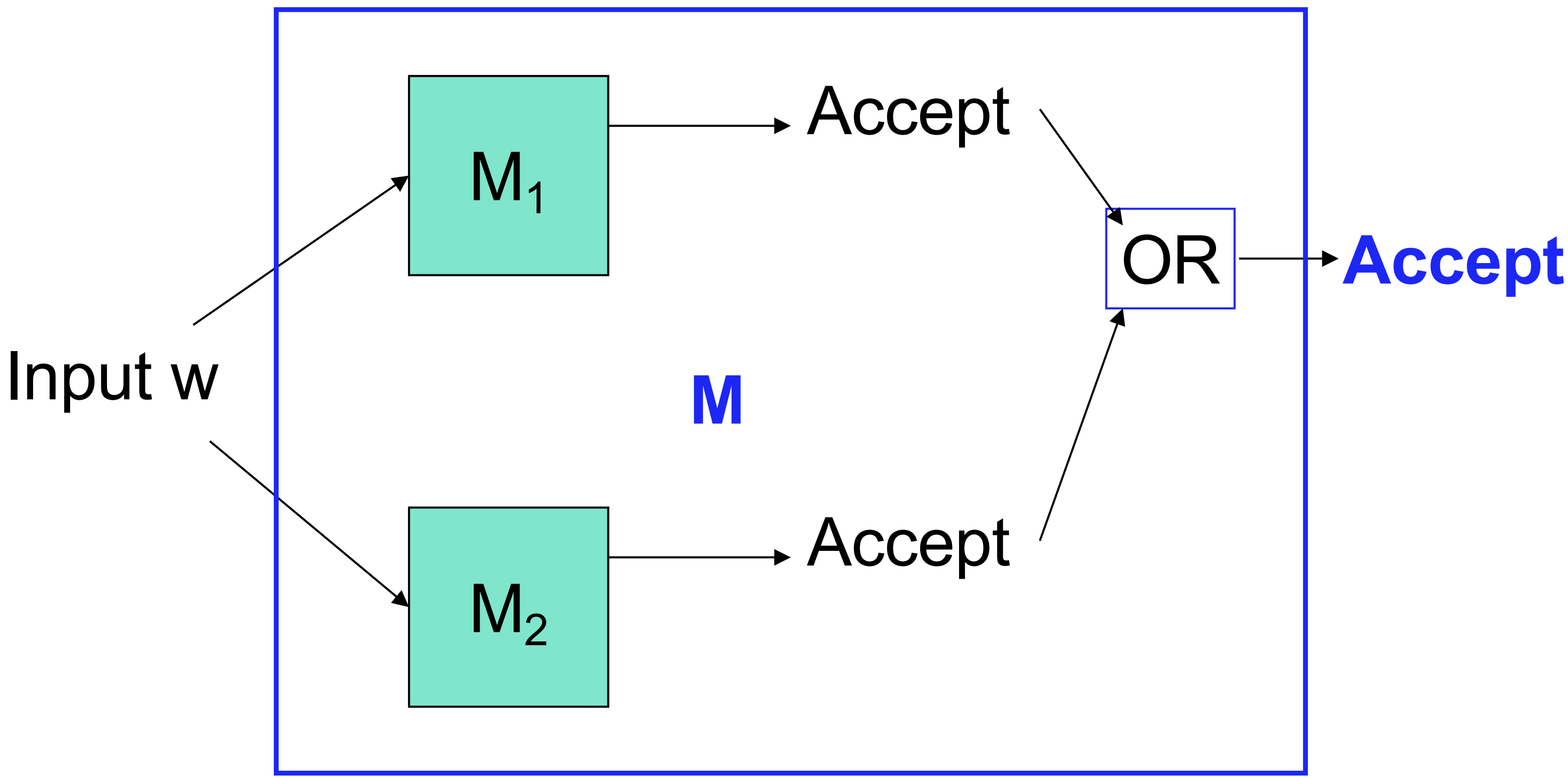
Algorithm/Picture of Union for Recursive Sets

M must halt on all inputs:
accepts if w is in either, rejects if w not in either



Picture of Union of RE Sets

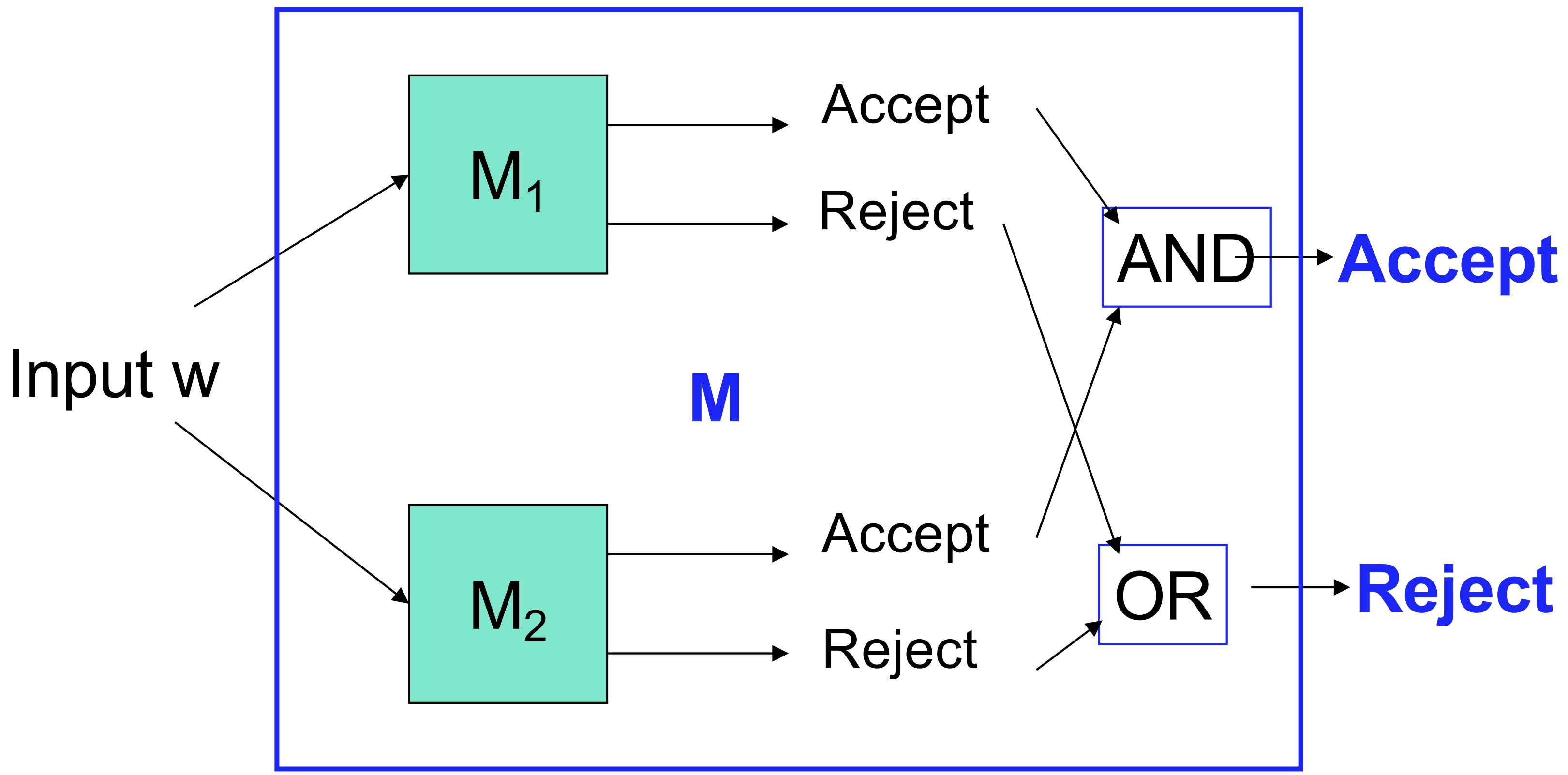
M must halt and accept if w is in either language, else it may reject and halt or may not halt



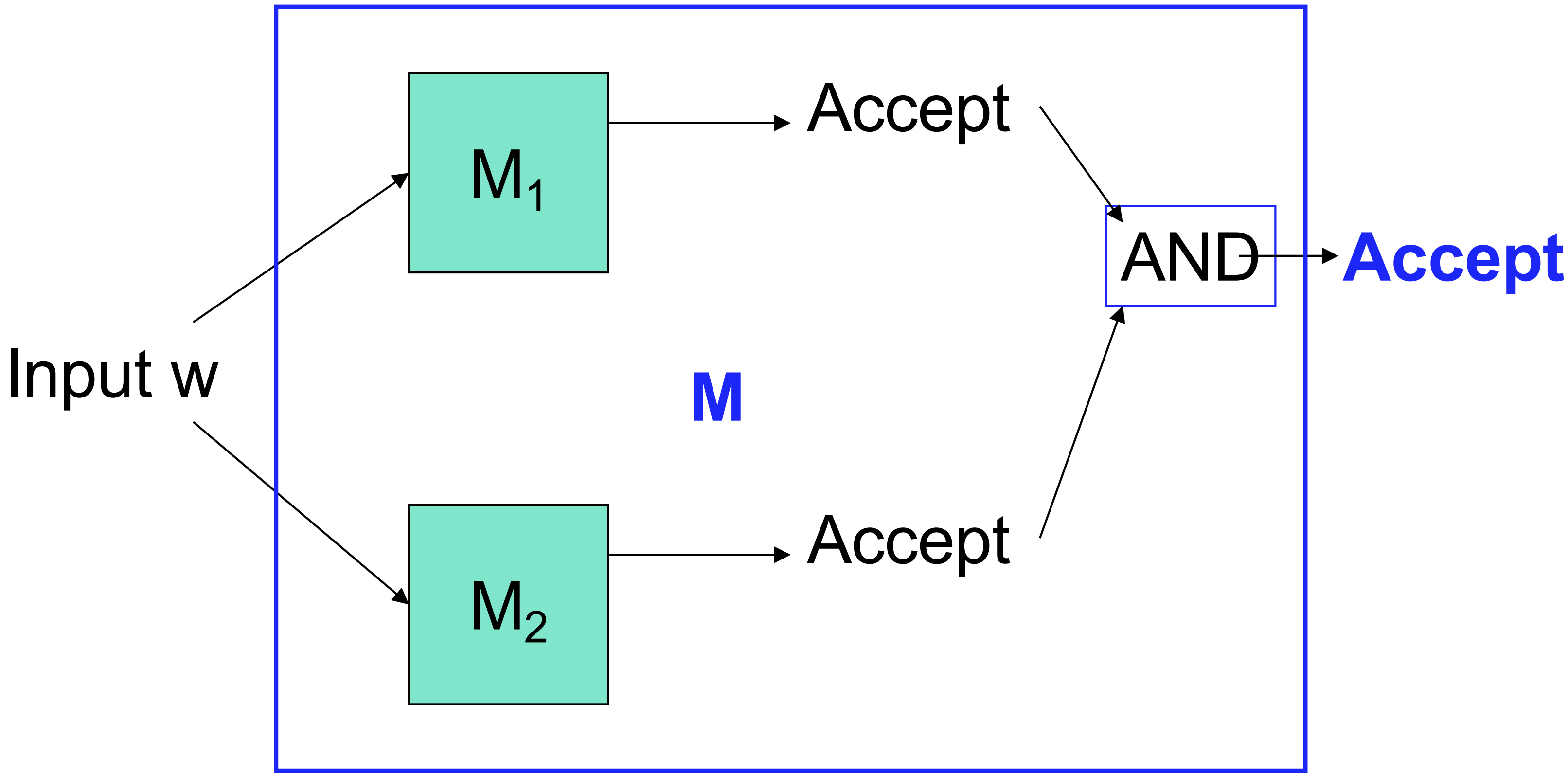
Closure under other set operations

- Recursive Languages are closed under
  - Union, Intersection, Concatenation, Star Closure
  - Complementation. Set difference
  - Reversal
  - Inverse Homomorphism
- Recursively Enumerable (RE) languages are closed under
  - Union, Intersection, Concatenation, Star Closure
  - Reversal
  - Homomorphism
  - Inverse Homomorphism

Intersection of Recursive Sets – Same Idea

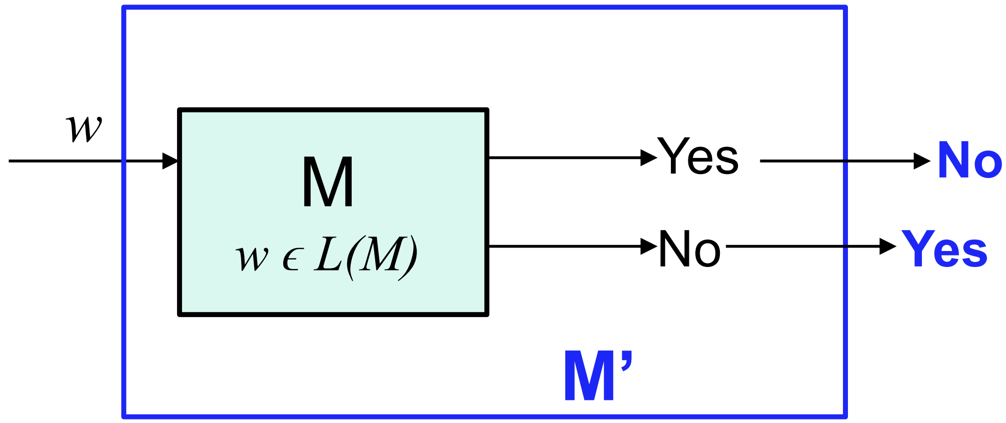


Intersection of RE Sets



Observe: if w is in the intersection then both
machines will accept and halt on w =>
This machine M will halt and accept w

Complement of Recursive Languages



Set Difference, Complement

- Recursive languages: both TM's will eventually halt.
- Accept if M_1 accepts and M_2 does not.
  - *Corollary*: Recursive languages are closed under complementation.
- RE Languages: can't do it; M_2 may never halt, so you can't be sure input is in the difference.

Concatenation of RE Languages

- Let $L_1 = L(M_1)$ and $L_2 = L(M_2)$.
- Assume M_1 and M_2 are single-semi-infinite-tape TM's.
- Construct 2-tape Nondeterministic TM M:
  1. Guess a break in input $w = xy$
  2. Move y to second tape.
  3. Simulate M_1 on x, M_2 on y.
  4. **Accept if both accept**.

Concatenation of Recursive Languages

- Can't use a NTM.
- Systematically try each break w = xy.
- M_1 and M_2 will eventually halt for each break.
- Accept if both accept for any one break.
- Reject if all breaks tried and none lead to acceptance.

Star Closure

- Same ideas work for each case.
- RE: guess many breaks, accept if M_1 accepts each piece.
- Recursive: systematically try all ways to break input into some number of pieces.

Reversal

- Start by reversing the input.
- Then simulate TM for L to accept w if and only w^R is in L.
- Works for either Recursive or RE languages.

Inverse Homomorphism

- Apply h to input w.
- Simulate TM for L on h(w).
- Accept w iff h(w) is in L.
- Works for Recursive or RE.

Homomorphism/RE

- Let L = L(M_1).
- Design NTM M to take input w and guess an x such that h(x) = w.
- M accepts whenever M_1 accepts x.
- Note: won't work for Recursive languages.