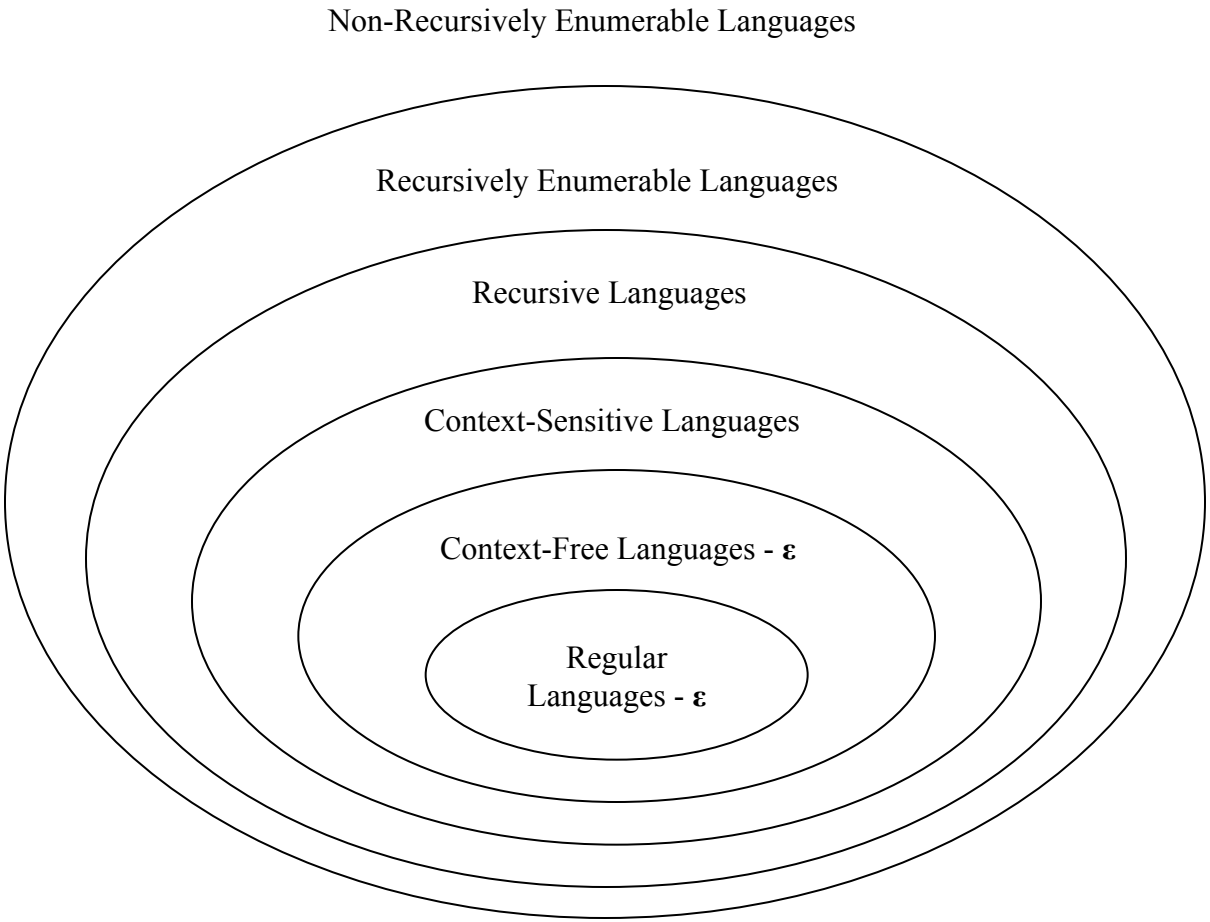
Decidability & Undecidability

- **Recall the Hierarchy:**



- **Observation:** Let L be an r.e. language. Then there is an infinite list M_0, M_1, … of TMs such that L = L(M_i).

- **Question:** Let L be a **recursive** language, and M_0, M_1, … a list of all TMs such that L = L(M_i), and choose any i>=0. Does M_i always halt?

- **Answer:** Maybe, maybe not, but *at least one in the list does*.

- **Question:** Let L be a **recursive enumerable** language, and M_0, M_1, … a list of all TMs such that L = L(M_i), and choose any i>=0. Does M_i always halt?

- **Answer:** Maybe, maybe not. Depending on L, none might halt or some may halt.

  - If L is also recursive then L is recursively enumerable, *recursive is subset of r.e.*

- **Question:** Let L be a r.e. language that is not recursive (L is in r.e. – r), and M_0, M_1, … a list of all TMs such that L = L(M_i), and choose any i>=0. Does M_i always halt?

- **Answer:** No! If it did, then L would not be in r.e. – r, it would be recursive.

L is Recursively enumerable:

TM exist: M_0, M_1, ...

They accept string in L, and do not accept any string outside L

L is Recursive:

at least one TM halts on L and on $\sum$-L, others may or may not*

L is Recursively enumerable but not Recursive:

TM exist: M_0, M_1, ...

but <u>none</u> halts on <u>all</u> x in $\sum$-L*

M_0 goes on infinite loop on a string p in $\sum$-L, while M_1 on q in $\sum$*-L*

However, each correct TM accepts each string in L, and none in $\sum$-L*

L is not R.E*:*

no TM exists

- **Let *M* be a TM.**
  - Question: Is L(M) r.e.?
  - Answer: Yes! By definition it is!
  - Question: Is L(M) recursive?
  - Answer: Don’t know, we don’t have enough information.
  - Question: Is L(M) in r.e – r?
  - Answer: Don’t know, we don’t have enough information.

- **Let *M* be a TM that halts on all inputs:**
  - Question: Is L(M) recursively enumerable?
  - Answer: Yes! By definition it is!
  - Question: Is L(M) recursive?
  - Answer: Yes! By definition it is!
  - Question: Is L(M) in r.e – r?
  - Answer: No! It can't be. Since *M* always halts, L(M) is recursive.

- **Let *M* be a TM.**
  - As noted previously, L(M) is recursively enumerable, but may or may not be recursive.
  - Question: Suppose, we know L(M) is recursive. Does that mean *M* always halts?
  - Answer: Not necessarily. However, some TM *M'* must exist such that L(M') = L(M) and *M'* always halts.
  - Question: Suppose that L(M) is in r.e. – r. Does *M* always halt?
  - Answer: No! If it did then L(M) would be recursive and therefore not in r.e. – r.

- **Let *M* be a TM, and suppose that *M* loops forever on some string *x*.**

  - Question: Is L(M) recursively enumerable?
  - Answer: Yes! By definition it is. But, obviously *x* is not in L(M).

  - Question: Is L(M) recursive?
  - Answer: Don't know. Although *M* doesn't always halt, some other TM *M'* may exist such that L(M') = L(M) and *M'* always halts.

  - Question: Is L(M) in r.e. – r?
  - Answer: Don't know.

    May be another *M'* will halt on *x*, and on all strings! May be no TM for this L(M) does halt on all strings! We just do not know!

Modifications of the Basic TM Model

- **Other (Extended) TM Models:**
  - One-way infinite tapes
  - Multiple tapes and tape heads
  - Non-Deterministic TMs
  - Multi-Dimensional TMs (n-dimensional tape)
  - Multi-Heads
  - Multiple tracks

All of these extensions are equivalent to the basic DTM model

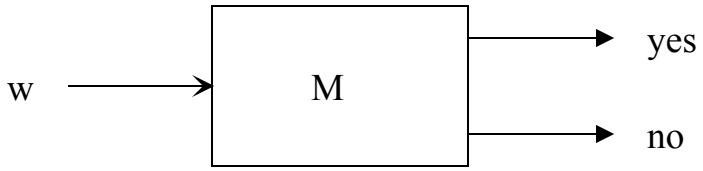
Closure Properties for Recursive and Recursively Enumerable Languages

- **TMs model General Purpose (GP) Computers:**
  - If a TM can do it, so can a GP computer
  - If a GP computer can do it, then so can a TM

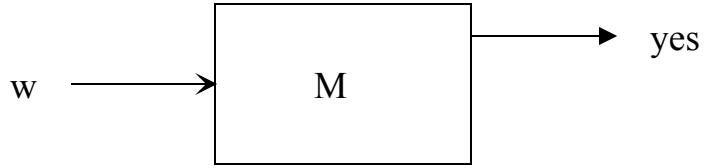
  If you want to know if a TM can do X, then some equivalent question are:
  - *Can a general purpose computer do X?*
  - *Can a C/C++/Java/etc. program be written to do X?*

  For example, is a language L recursive?
  - *Can a C/C++/Java/etc. program be written that always halts and accepts L?*

- **TM Block Diagrams:**
  - If *L* is a recursive language, then a TM *M* that accepts *L* and always halts can be pictorially represented by a "chip" or "box" that has one input and two outputs.



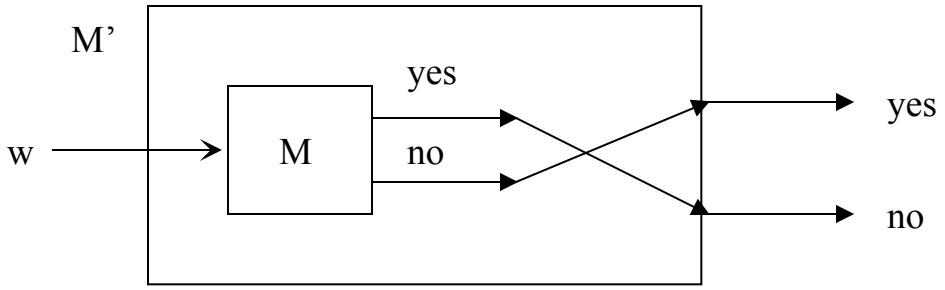
  - If *L* is a recursively enumerable language, then a TM *M* that accepts *L* can be pictorially represented by a "box" that has one output.



  - Conceivably, *M* could be provided with an output for "no," but this output cannot be counted on. Consequently, we simply ignore it.

- **Theorem 1:** The recursive languages are closed with respect to complementation, i.e., if *L* is a recursive language, then so is $\overline{L} = \Sigma^* - L$

- **Proof:** Let *M* be a TM such that L = L(M) and *M* always halts. Construct TM *M'* as follows:



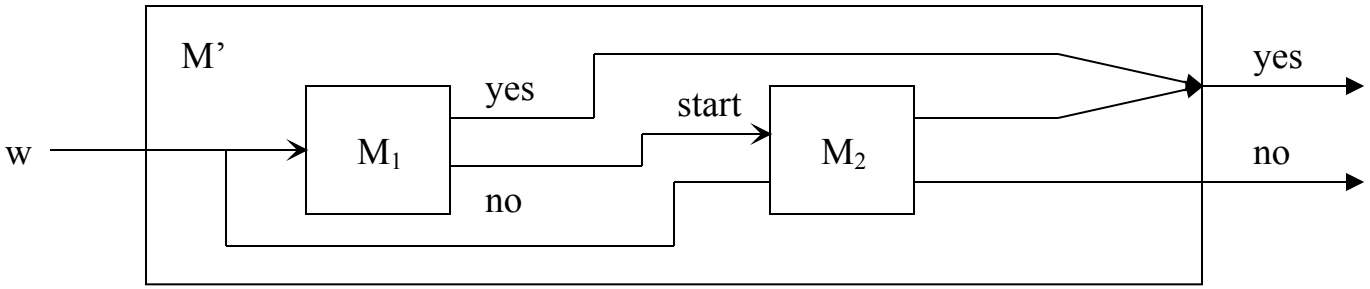
- **Note That:**
  - *M'* accepts iff *M* does not
  - *M'* always halts since *M* always halts

  From this it follows that the complement of *L* is recursive.

- **Question:** How is the construction achieved? Do we simply complement the final states in the TM? No! A string in *L* could end up in the complement of *L*.
  - Suppose q_5 is an accepting state in *M*, but q_0 is not.
  - If we simply complemented the final and non-final states, then q_0 would be an accepting state in *M'* but q_5 would not.
  - Since q_0 is an accepting state, by definition all strings are accepted by *M'*

- **Theorem 2:** The recursive languages are closed with respect to union, i.e., if L_1 and L_2 are recursive languages, then so is $L_3 = L_1 \cup L_2$

- **Proof:** Let M_1 and M_2 be TMs such that $L_1 = L(M_1)$ and $L_2 = L(M_2)$ and M_1 and M_2 always halts. Construct TM M' as follows:

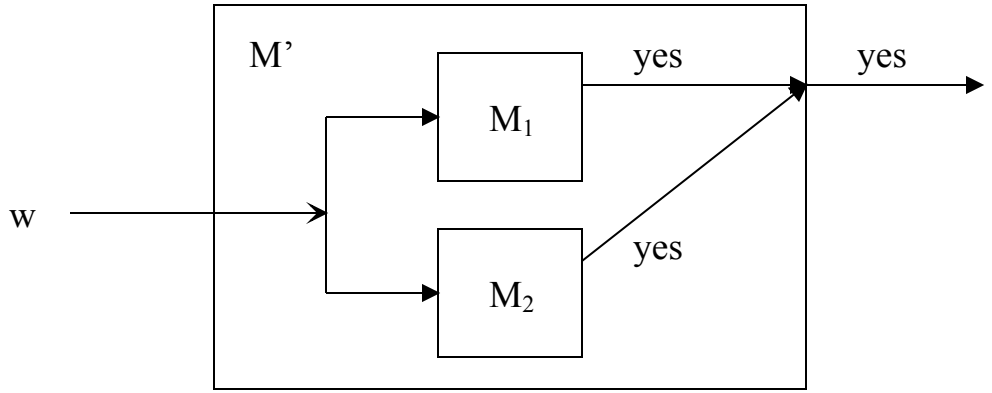


- **Note That:**
  - $L(M') = L(M_1) \cup L(M_2)$
    - $L(M')$ is a subset of $L(M_1) \cup L(M_2)$
    - $L(M_1) \cup L(M_2)$ is a subset of $L(M')$
  - M' always halts since M_1 and M_2 always halt

It follows from this that $L_3 = L_1 \cup L_2$ is recursive.

- **Theorem 3:** The *recursive enumerable languages* are closed with respect to union, i.e., if L_1 and L_2 are recursively enumerable languages, then so is $L_3 = L_1 \cup L_2$

- **Proof:** Let M_1 and M_2 be TMs such that $L_1 = L(M_1)$ and $L_2 = L(M_2)$. Construct M' as follows:

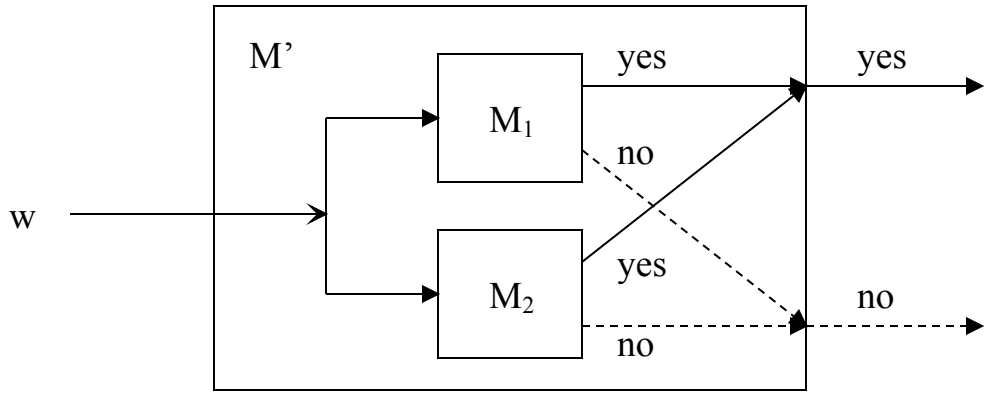


- **Note That:**
  - $L(M') = L(M_1) \cup L(M_2)$
    - $L(M')$ is a subset of $L(M_1) \cup L(M_2)$
    - $L(M_1) \cup L(M_2)$ is a subset of $L(M')$
  - M' halts and accepts iff M_1 or M_2 halts and accepts

It follows from this that $L_3 = L_1 \cup L_2$ is recursively enumerable.

- ***Question:*** *How do you run two TMs in parallel?*

- Suppose, M_1 and M_2 had outputs for "no" in the previous construction, and these were transferred to the "no" output for *M'*



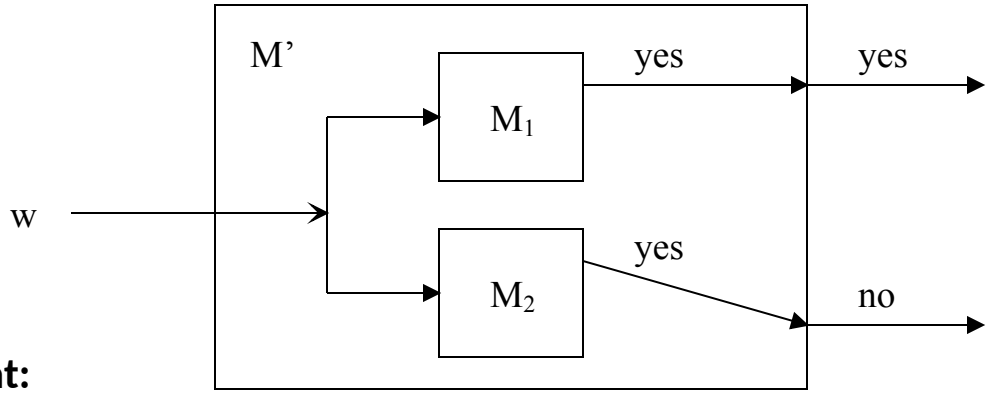
- **Question:** What would happen if *w* is in L(M_1) but not in L(M_2)?

- **Answer:** You could get two outputs – one "yes" and one "no."

  - At least M_1 will halt and answer accept, M_2 may or may not halt.
  - As before, for the sake of convenience the "no" output will be ignored.

- **Theorem 4:** If *L* and $\overline{L}$ are both recursively enumerable then *L* (and therefore $\overline{L}$) is recursive.

- **Proof:** Let M_1 and M_2 be TMs such that L = L(M_1) and $\neq \overline{L}$ L(M_2). Construct *M'* as follows:



- **Note That:**
  - L(M') = L
    - L(M') is a subset of *L*
    - *L* is a subset of L(M')
  - M' is TM for L
  - *M'* always halts since <u>either</u> M_1 <u>or</u> M_2 halts for any given string
  - M' shows that L is recursive

  It follows from this that *L* (and therefore its' complement) is recursive.

  So, $\overline{L}$ is also recursive (we proved it before).

- **Corollary of Thm 4:** Let *L* be a subset of Σ*. Then one of the following must be true:

  - Both *L* and $\overline{L}$ are recursive.
  - One of *L* and $\overline{L}$ is recursively enumerable but not recursive, and the other is not recursively enumerable, or
  - Neither *L* nor $\overline{L}$ is recursively enumerable

  - *In other words, it is impossible to have both L and* $\overline{L}$ *r.e. but not recursive*

The Halting Problem - Background

- **Definition:** A decision problem is a problem having a yes/no answer (that one presumably wants to solve with a computer). Typically, there is a list of parameters on which the problem is based.
  - Given a list of numbers, is that list sorted?
  - Given a number x, is x even?
  - Given a C program, does that C program contain any syntax errors?
  - Given a TM (or C program), does that TM contain an infinite loop?

  From a practical perspective, many decision problems do not seem all that interesting. However, from a theoretical perspective they are for the following two reasons:
  - Decision problems are more convenient/easier to work with when proving complexity results.
  - Non-decision *counter-parts* can always be created & are typically at least as difficult to solve.

- **Notes:**
  - The following terms and phrases are analogous:

Algorithm		-	A halting TM program
Decision Problem	-	A language	*(will show shortly)*
(un)Decidable	**-**	**(non)Recursive**	