General Turing Reductions:

- P **Turing reduces** to Q if there exists an algorithm for P that uses an algorithm for Q as a "black box". This is denoted as $P \leq_T Q$.

Halting Problem:

- Denoted as H.
- H = {⟨M, x⟩ | TM M halts on input x}
  H accepts the encodings of M and x if M halts on x, and rejects the encodings otherwise.
- **Theorem 4.1:** H is
  a. recognizable but
  b. not decidable.

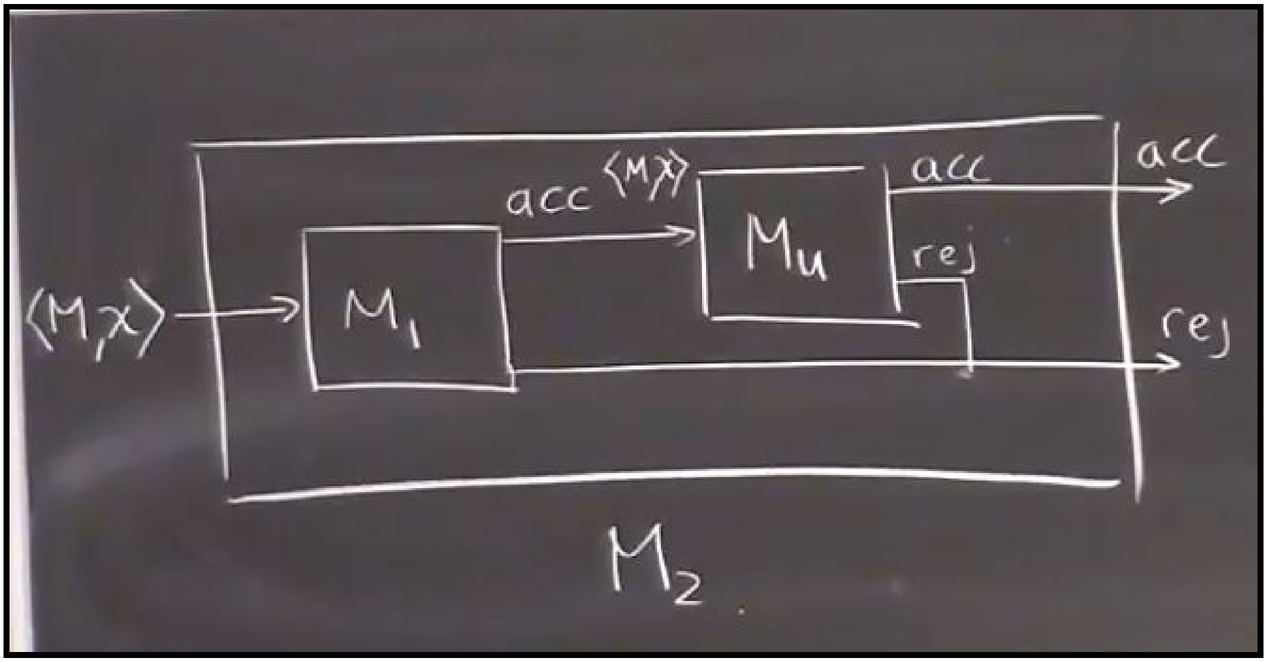
  Proof of a):
  Use M_u to simulate M on input x.
  If M_u halts (either accepts or rejects), then we say yes.
  If M_u doesn't halt, then it's fine because H is a recognizer, not a decider.

  Proof of b):
  We will show that U ≤ H.
  U is the universal language. In theorem 3.5, we proved that U is recognizable but not decidable.
  Given an H-decider TM M_1, we will construct a U-decider TM M_2.
  M_2 on input ⟨M, x⟩ does the following:
  1. Run M_1 on ⟨M, x⟩
  2. If M_1 accepts, then
  3. Run M_u on ⟨M, x⟩
  4. If M_u accepts ⟨M, x⟩, then M_2 accepts
  5. Else, M_2 rejects
  6. Else, M_2 reject

  M_2 accepts U.
  First, it runs M_1 on ⟨M, x⟩.
  If M_1 accepts, meaning that it halts on ⟨M, x⟩, then we run M_u on ⟨M, x⟩.
  If M_u accepts ⟨M, x⟩, then M_2 accepts ⟨M, x⟩.
  If M_u doesn't accept ⟨M, x⟩, then M_2 rejects ⟨M, x⟩.
  If M_1 doesn't accept ⟨M, x⟩, that means M_1 doesn't halt on ⟨M, x⟩, so M_2 rejects ⟨M, x⟩.

  However, we proved in theorem 3.5 that U is not decidable, so M_2 doesn't exist.
  Since M_2 relies on the existence of M_1, therefore, M_1 doesn't exist.
  Therefore, H is undecidable.

Here's a diagram of the proof:



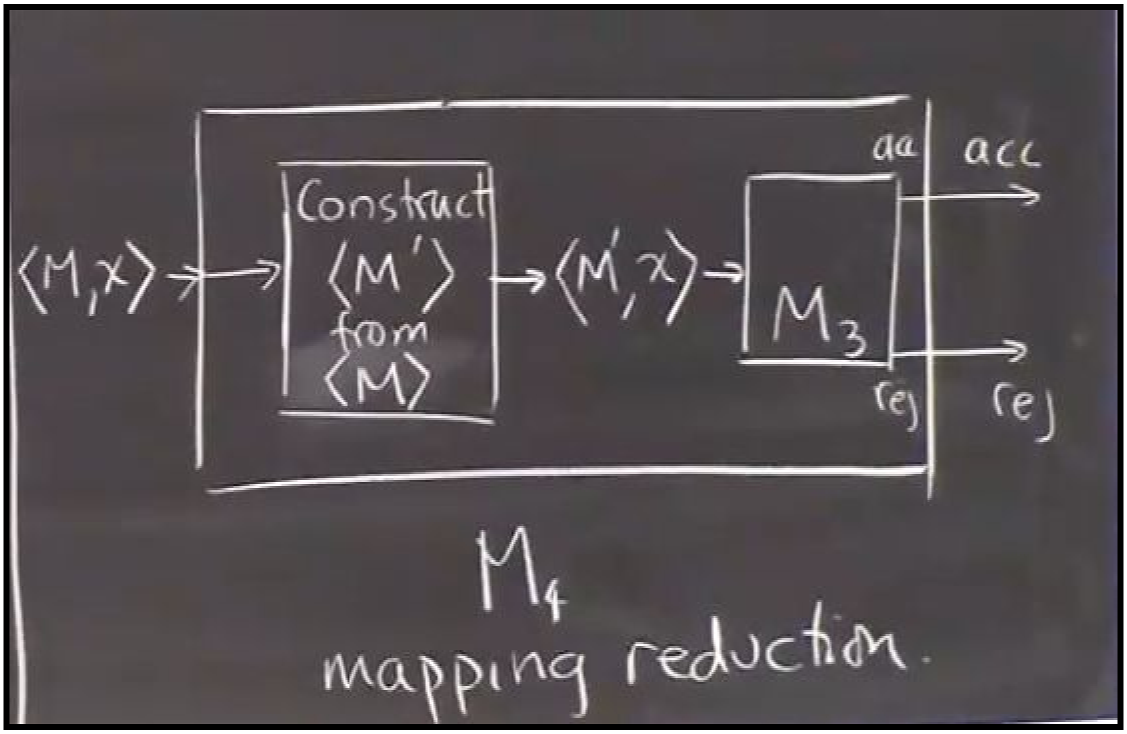
Alternative Proof of b):

Given an H-decider M_3, we can construct a U-decider M_4 as follows:
M_4 on input ⟨M, x⟩ does the following:

1. Modify M to M' by changing every transition of M to the reject state into an infinite loop.
   We know that M either accepts, rejects or loops on x.
   If M accepts x, then M' accepts x.
   If M rejects or loops on x, then M' loops on x.
2. Run M_3 on ⟨M', x⟩.
3. If M_3 accepts, then M_4 accepts.
4. Else, M_4 rejects.

M_4 accepts ⟨M, x⟩
↔ M_3 accepts ⟨M', x⟩
↔ M' halts on x
↔ M accepts x
Therefore, M_4 is a U-decider.
However, this contradicts theorem 3.5, which says that U is not decidable.
Therefore, we could not have been given an M_3 that solves the halting problem.

Here's a diagram of the proof:



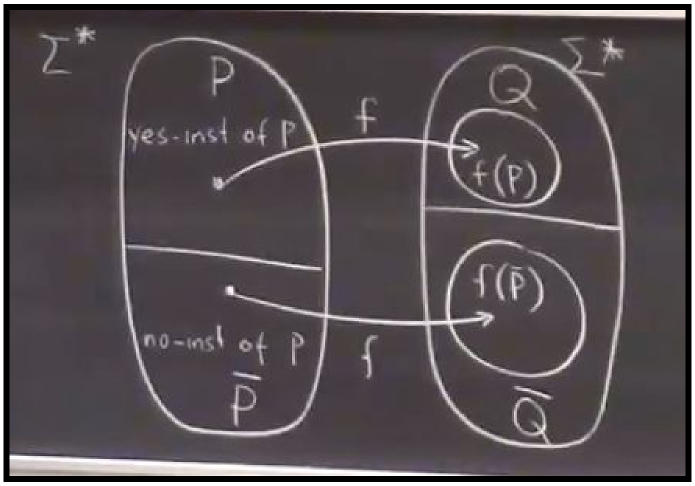
- **Corollary 4.2:** ¬H, the complement of H, is unrecognizable.
  ¬H = {⟨M, x⟩ | M doesn't halt on x}

  Proof:
  Suppose by contradiction that ¬H is recognizable.
  Recall theorem 3.6 "If L and ¬L, the complement of L, are both recognizable, then L and ¬L are decidable."
  Since both H and ¬H are recognizable, then both H and ¬H are decidable.
  However, we know that H is not decidable, which is a contradiction.
  Hence, ¬H is unrecognizable.
- If X ≤ Y and X is undecidable, then Y is also undecidable.
  However, if X ≤ Y and Y is undecidable, it doesn't tell us if X is undecidable or not.
  Note: The direction in which we are reducing things is very important.
  E.g.
  When we did U ≤ H, since we knew that U is undecidable, we could prove that H is undecidable.
  However, if we did H ≤ U, we know that U is undecidable, but we don't know if H is undecidable. We can't use this to prove that H is undecidable.
- If X ≤ Y and Y is decidable, then X is also decidable.

Mapping Reductions:

- **Definition:** Let P and Q ⊆ Σ^* be languages. P is **mapping-reducible** to Q, denoted as P $\leq_m$ Q, iff there exists a computable function, f : $\Sigma^* \to \Sigma^*$, such that x ∈ P iff f(x) ∈ Q.
  Note: The function, f, does not have to be, and is usually not, onto.
  Note: The function, f, must be computable.
  To demonstrate a computable function, we will typically write a little program or describe in English how to perform the transformation that f is supposed to do.

Note: f maps yes-instances of P to yes-instances of Q and no-instances of P to no-instances of Q.
Here is a diagram to show the definition of mapping-reducible:



Here, f maps the Yes-instances of P to a subset of the Yes-instances of Q and maps the No-instances of P to a subset of the No-instances of Q.

- E.g. Suppose that
  A = {x | x is an even integer}
  B = {x | x is an odd integer}
  Then the function f(x) = x + 1 is a mapping reduction from A to B.
  Notice that:
  $x \in A \leftrightarrow$ x is even
  $\leftrightarrow x + 1$ is odd
  $\leftrightarrow x + 1 \in B$
  $\leftrightarrow f(x) \in B$
- All the reductions we've seen so far, with one exception, are mapping reductions.
  1. **First Reduction: Reduced ¬D (D complement) to U (Universal language)**
     - ¬D = {⟨M⟩ | M accepts ⟨M⟩}
     - f: ⟨M⟩ → ⟨M, ⟨M⟩⟩
       Here is a description of f:
       - Take the encoding of M.
       - Make a pair of itself and another encoding of M in the following way: ⟨M⟩###⟨M⟩ (The ### is used as a separator.)
  2. **Second Reduction: Reduce U to H (The Halting Problem)**
     - **Note:** This is for the "Alternative Proof of b)"
     - Given ⟨M, x⟩ we constructed ⟨M', x⟩ such that M accepts x iff M' halts on x.
       M accepts x simply means ⟨M, x⟩∈U and M' halts on x simply means ⟨M', x⟩∈H.
       So, I mapped ⟨M, x⟩ to ⟨M', x⟩ such that Yes-instances go to Yes-instances and No-instances go to No-instances.
     - **Note:** The first proof we did to prove that U reduces to H is not a mapping reduction. The difference between the first and second proof is that with the first proof, we're taking the input, ⟨M, x⟩, and running it through 2 "black boxes", M_1 and M_u. Furthermore, after running the input through the first "black box", M_1, there's a possibility that we're changing its output by running the output through the second "black box", M_u.

With the second proof, we're transforming ⟨M, x⟩ to ⟨M', x⟩, this is our function, and we're only running it through 1 TM, M_3. In the second proof, we're using M_3 in a very restricted way. We are only making 1 call to the "black box" and we're using the output of the "black box" as it is, we can't change it.

- Hence, the first proof is a **Turing reduction** while the second proof is a **mapping reduction**.

- **Theorem 4.3:** Suppose that $P \leq_m Q$. If Q is decidable, then P is decidable. If P is undecidable, then Q is undecidable.

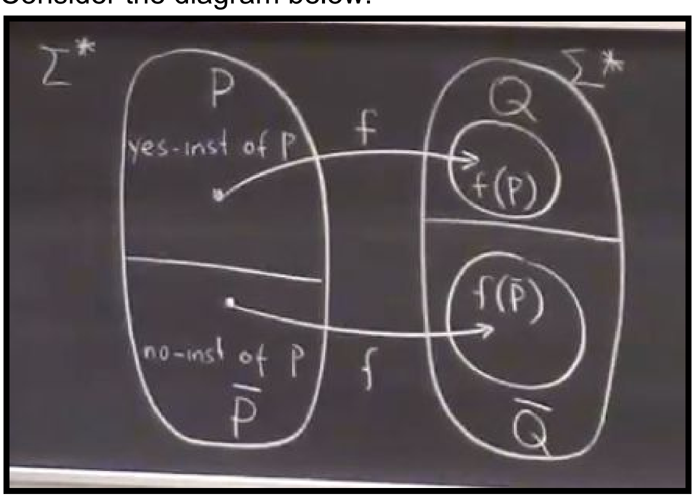
  Proof of "If P is undecidable, then Q is undecidable":
  Assume that $P \leq_m Q$ and P is undecidable.
  Suppose for contradiction that Q is decidable.
  Let D_Q be a decider for Q.
  Since $P \leq_m Q$, there exists a computable function, f, such that $x \in P$ iff $f(x) \in Q$.
  Then, the following algorithm is a decider for P:
  D_P on input "x" does the following:
  1. Computes f(x)
  2. Run D_Q on f(x).
  3. If D_Q accepts, then D_P accepts.
  4. Else, D_P rejects.

  D_P halts on all inputs, so it's a decider.
  D_P decides P because it accepts x iff D_Q accepts f(x).
  D_Q accepts f(x) iff $f(x) \in Q$, because D_Q is a decider for Q.
  $f(x) \in Q$ iff $x \in P$, because f is a mapping reduction of P to Q.
  However, this contradicts our supposition that P is undecidable.
  Hence, Q is undecidable.

- **Theorem 4.4:** If $P \leq_m Q$ and Q is recognizable, then P is recognizable. If P is unrecognizable, then Q is unrecognizable.

- **Theorem 4.5:** If $P \leq_m Q$, then $\neg P \leq_m \neg Q$, where $\neg P$ is the complement of P and $\neg Q$ is the complement of Q.

  Proof:
  Consider the diagram below.



We know that f maps the Yes-instances of P to the Yes-instances of Q and the No-instances of P to the No-instances of Q.
However,

- The No-instances of P are the same as the Yes-instances of ¬P.
- The No-instances of Q are the same as the Yes-instances of ¬Q.
- The Yes-instances of P are the same as the No-instances of ¬P.
- The Yes-instances of Q are the same as the No-instances of ¬Q.

Hence, we can use the same function, f, as the computable function for $\neg P \leq_m \neg Q$.

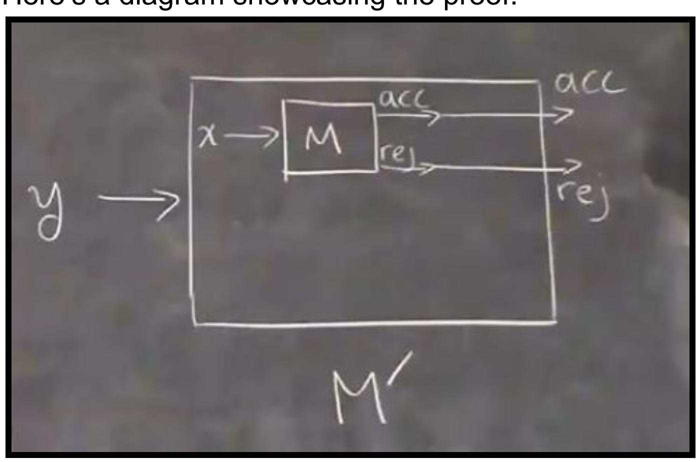
- **Theorem 4.6:** If $P \leq_m Q$ and $Q \leq_m R$, then $P \leq_m R$.

Examples of Reductions:

- To prove that a language P is unrecognizable or undecidable, it suffices to prove that $U \leq_m P$, for undecidable, and $\neg U \leq_m P$, for unrecognizable. This is by theorem 4.3 and 3.4.
- **Theorem 4.7:** Consider the following language, E = {⟨M⟩ | L(m) = Ø}. E is unrecognizable.

  Proof:
  It suffices to prove that $\neg U \leq_m E$.
  Given ⟨M,x⟩, which is the input to ¬U, we want to construct ⟨M'⟩, which is the input to E, such that M does not accept x iff L(M') = Ø.
  I.e. ⟨M,x⟩ ∈ ¬U iff ⟨M'⟩ ∈ E.
  We can build M' such that if M doesn't accept x, M' accepts no string, and if M accepts x, M' accepts every string.
  f on input ⟨M,x⟩ does the following:
  1. Define a machine M' that does the following on input y:
     a. Run M on x
     b. If M accepts, then M' accepts y.
     c. Else, M' rejects y.
  2. Return ⟨M'⟩

  Here's a diagram showcasing the proof.



  If M does not accept x, then L(M') = Ø.
  If M accepts x, then L(M') = Σ^*.

Claim: f is a mapping reduction of ¬U to E.
Proof:
To prove that f is a mapping reduction of ¬U to E, we need to verify that
⟨M,x⟩ ∈ ¬U iff ⟨M'⟩ ∈ E.
(=>) If ⟨M,x⟩ ∈ ¬U
→ M does not accept x. (M either loops on x or M rejects x.)
→ M' accepts no input.
→ L(M') = Ø
→ ⟨M'⟩ ∈ E

(<=) If ⟨M,x⟩ ∉ ¬U
→ M accepts x.
→ M' accepts all inputs.
→ L(M') = $\sum^*$ ≠ Ø
→ ⟨M'⟩ ∉ E

- **Theorem 4.8:** ¬E, the complement of E, is
    a. undecidable, but
    b. recognizable

  ¬E = {⟨M⟩| L(M) ≠ Ø}

Proof of a):
Suppose for contradiction that ¬E is decidable.
Then, based on theorem 3.3, which states that
"If L is a decidable language, then its complement is also decidable.
I.e. The set of decidable languages is closed under complementation.",
then E is also decidable.
However, we just proved in theorem 4.7 that E is undecidable, which is a contradiction.
Hence, ¬E is undecidable.

Proof of b):
The idea is to dovetail through all pairs (i,j). When visiting pair (i,j), run M on the i^{th} input for j steps. If it accepts, then we accept. Otherwise, visit the next pair.
If M doesn't accept or reject the i^{th} input for j steps, we simply continue the dovetailing process. This is fine because as a recognizer, it doesn't need to halt.
Note: The reason why you can't simply go down each input is because there might be an input that loops forever. Then, your machine would be stuck.

Another Proof of b):
A NTM recognizes ¬E on input ⟨M⟩ as follows:
1. Nondeterministically guess a string x.
2. Use M_u, the universal TM, to run M on x.
3. If M accepts, accept.
4. Since there's a NTM that recognizes ¬E, there's also a TM that recognizes ¬E.

- **Theorem 4.9:** Consider the following language, REG = {⟨M⟩ | L(M) is regular}. REG is undecidable.

  Proof:
  It suffices to prove that U $\leq_M$ REG.
  Given an input, ⟨M, x⟩ to U, we want to construct a machine ⟨M'⟩, which is an input to REG such that M accepts x iff L(M') is regular.
  If M accepts x, then M' accepts a regular language.
  If M does not accept x, then M does not accept a regular language.
  f on input ⟨M, x⟩ does the following:
  1. Define M' which on input y does the following:
     a. If $y=0^n1^n$, then accept
     b. Else, run M on x.
     c. If M accepts x, then M' accepts y.
     d. Else, M' rejects y.
  2. Return ⟨M'⟩

  Now, we need to verify that ⟨M, x⟩ ∈ U iff ⟨M'⟩ ∈ REG.
  (=>)
  If ⟨M, x⟩ ∈ U then
  → M accepts x.
  → M' accepts all inputs y. It does this in either line 1a. or line 1c. otherwise.
  → L(M') = Σ^*.
  → ⟨M'⟩ ∈ REG.

  (<=)
  If ⟨M, x⟩ ∉ U then
  → M does not accept x.
  → M' accepts all and only strings of the form 0^n1^n.
  → L(M') is not regular.
  → ⟨M'⟩ ∉ REG.
  We have shown that U $\leq_M$ REG, so REG is undecidable.

  Note: Since U $\leq_M$ REG, ¬U $\leq_M$ ¬REG. This is by theorem 4.5.
  Since ¬U $\leq_M$ ¬REG, ¬REG is unrecognizable.
  Note: U $\leq_M$ ¬REG, which means that ¬U $\leq_M$ REG.
  Since ¬U $\leq_M$ REG, REG is unrecognizable.

Picture of Recognizable and Decidable Languages: