More examples of mapping reductions:

- Consider the following set, HET = {⟨M⟩ | TM M halts on empty tape}.
  HET stands for Halts on Empty Tape. The input for HET is ε (the empty string).
- **Theorem 5.1:** HET is not decidable.

  Proof:
  It suffices to show that U $\leq_m$ HET.
  Given ⟨M,x⟩ to U, we want to construct ⟨M'⟩ to HET, such that M accepts x iff M' halts on empty tape.
  If M accepts x, then M' should halt on empty tape.
  If M does not accept x, then M' should not halt on empty tape.

  f on input ⟨M,x⟩ does the following:
  1. Define M'. M' on input y does the following:
     a. Runs M on x.
     b. If M accepts, then M' accepts y.
     c. Else, loop.
  2. Returns ⟨M'⟩

  If M accepts x, then M' accepts Σ^* (everything).
  In particular, M' accepts ε.
  This means that M' halts on empty tape.
  This means that ⟨M'⟩ ∈ HET.

  If M does not accept x, then M' always loops.
  M' could loop at 2 places:
  1. Line 1a. "Runs M on x". If M does not accept x because it loops, then M' loops here.
  2. Line 1c. "Else, loop" If M does not accept x because it rejects it, then M' loops here.

  In particular, M' does not halt on empty tape.
  Therefore, ⟨M'⟩ ∉ HET.

  Note: HET is recognizable.
- **Theorem 5.2:** Let ODD = {⟨M⟩ | L(M) is finite and |L(M)| is odd}. ODD is not recognizable.

  Proof:
  Suffices to prove that ¬U $\leq_m$ ODD.
  Given ⟨M, x⟩ to ¬U, we want to construct ⟨M'⟩ to ODD, such that M does not accept x iff L(M') is finite and |L(M)| is odd.
  If M does not accept x, then L(M') is finite and |L(M)| is odd.
  If M accepts x, then L(M') is infinite or |L(M)| is even.

f on input ⟨M, x⟩ does the following:

1. Define M'. M' on input y does the following:
    a. If y = 010, then accept.
    b. Run M on x.
    c. If M accepts x, then M' accepts anything. (M' accepts an infinite number of strings.)
    d. Else, M' rejects y.
2. Return ⟨M'⟩

If M does not accept x, then M' will either accept {010} (line 1a.) or it will reject everything. Hence, the only string L(M') can be is {010}. Therefore, ⟨M'⟩ ∈ ODD.

If M accepts x, L(M') = $\sum^*$, which is infinite. Therefore, ⟨M'⟩ ∉ ODD.

- **Theorem 5.3:** Let FIN = {⟨M⟩ | L(M) is finite} and let INF = ¬FIN.
  I.e. INF is the complement of FIN.
  I.e. INF = {⟨M⟩ | L(M) is infinite}
  Both FIN and INF are not recognizable.

  Proof:
  It suffices to show that
    a. U $\leq_m$ FIN
    b. U $\leq_m$ INF

  Note: This is the reason why we can use U instead of ¬U:
  Recall from theorem 4.5 that "If P $\leq_m$ Q, then ¬P $\leq_m$ ¬Q, where ¬P is the complement of P and ¬Q is the complement of Q."
  Since we can show that ¬U $\leq_m$ INF, we can show that ¬(¬U) $\leq_m$ ¬INF or U $\leq_m$ FIN.
  Similarly, since we can show that ¬U $\leq_m$ FIN, we can show that ¬(¬U) $\leq_m$ ¬FIN or U $\leq_m$ INF.

  Proof for INF:
  Given ⟨M, x⟩ to U, we want to construct ⟨M'⟩ to INF such that if M accepts x, then M' accepts an infinite language and if M doesn't accept x, then M' accepts a finite language.

  f on input ⟨M, x⟩ does the following:

  1. Define M'. M' on input y does the following:
      a. If y = Ø, then accept.
      b. Run M on x.
      c. If M accepts x, then M' accepts anything. (M' accepts an infinite number of strings.)
      d. Else, M' rejects y.
  2. Return ⟨M'⟩

  If M accepts x, then L(M') = $\sum^*$. Therefore, ⟨M'⟩ ∈ INF.
  If M does not accept x, then M' will either accept Ø (line 1a.) or it will reject everything.
  Therefore, ⟨M'⟩ ∉ INF.

Proof for FIN:
Given ⟨M, x⟩ to U, we want to construct ⟨M'⟩ to FIN such that if M accepts x, then M' accepts a finite language and if M doesn't accept x, then M' accepts an infinite language.

Note: There's a problem here. Normally, if M doesn't accept x, M' accepts a subset of what it would accept if M accepts x. (See below). However, in this case, if M doesn't accept x, M' accepts an infinite language while if M accepts x, M' accepts only a finite language.

f on input ⟨M, x⟩ does the following:

1. Define M'. M' on input y does the following:
    a. Run M on x for |y| steps.
    b. If M accepts x, in at most |y| steps, then M' reject.
    c. Else, M' accept.
2. Return ⟨M'⟩

Note: Because of line 1a., we don't have the problem that M runs on x forever.

If M accepts x, there exists a k, such that M accepts x after k steps. This means that M' accepts y if $|y| < k$ and rejects y if $|y| \geq k$. Hence, $L(M') = \{y \mid |y| < k\}$ ⟵ finite. Therefore, $\langle M' \rangle \in$ FIN.

If M does not accept x, then $\forall$ k, M does not accept x in k steps. Therefore, $\forall\ y \in \Sigma^*$, M' accepts y (in line 1c.). Therefore, $L(M') = \Sigma^*$ ⟵ infinite. Therefore, $\langle M' \rangle \notin$ FIN.

- Here is the general pattern for constructing "M' on input y does the following":
    1. M' might accept a certain input.
    2. Run M on x.
    3. If M accepts x, then maybe accept some more y's.
    4. Else, reject/loop

$$\boxed{L(M') = \begin{cases} \text{Y's accepted in line 1 or 3, if M accepts x.} \\ \text{Y's accepted in line 1, if M does not accept x.} \end{cases}}$$

Notice that if M does not accept x, then M' only accepts a subset of what it would accept if M accepts x.

- **Theorem 5.4:** Let EQUIV = {⟨M1, M2⟩ | L(M1) = L(M2)}. EQUIV is not recognizable.

  Proof:
  The standard approach is to show that $\neg U \leq_m$ EQUIV.
  Given ⟨M, x⟩ as to $\neg$U, we want to construct ⟨M1, M2⟩ to EQUIV, such that if M does not accept x then L(M1) ≡ L(M2) and if M accepts x, then L(M1) ≢ L(M2).

  We want to fix M1 to take a specific string, which will be 010.
  I.e. L(M1) = {010}
  I.e. M1 is a TM that only accepts the string 010 and nothing else.

$$L(M2) = \begin{cases} \{010\} & \text{if M doesn't accept x.} \\ \Sigma^* & \text{if M accepts x.} \end{cases}$$

  f on input ⟨M, x⟩ does the following:
  1. Define M1. M1 on input y does the following:
     a. If y = 010, then accept.
     b. Else, reject.
  2. Define M2. M2 on input z does the following:
     a. If z is 010, then accept.
     b. Run M on x.
     c. If M accepts, then M2 accepts.
     d. Else, M2 rejects.
  3. Returns ⟨M1, M2⟩

  Alternative Proof:
  Here, we will prove that $E \leq_m$ EQUIV, where E = {⟨M⟩ | L(m) = Ø}.
  ⟨M⟩ ∈ E iff ⟨M, $M_Ø$⟩ ∈ EQUIV, where $M_Ø$ is any TM that accepts no string.
  This proof would be easier to prove than the first proof.
- **Theorem 5.5:** Let SUBSET = {⟨M1, M2⟩ | L(M1) ⊆ L(M2)}. SUBSET is not recognizable.

  Proof:
  ⟨M⟩ ∈ E iff ⟨M, $M_Ø$⟩ ∈ SUBSET, where $M_Ø$ is any TM that accepts no string.

<u>Rice's Theorem:</u>

- **Theorem 5.6:** **Rice's theorem** states that if P is a nontrivial property of recognizable languages, then L_P is undecidable.
  L_P = {⟨M⟩ | L(M) ∈ P}
  L_P is the set of TM codes whose language has property P.
- A **trivial property** of a language is a property that all languages have or no languages have.

- A **nontrivial property** of a language is a property such that there is at least one language that satisfies the property and at least one language that does not.
- A **property** P of languages is simply a set of languages.
  E.g. The set of finite languages is a property.
  E.g. The set of infinite languages is a property.
  E.g. The set of finite languages that have an odd number of strings in them is a property.
- Some trivial properties are:
  - The empty set of languages. Denoted as Ø.
    Ø is a language which has no strings. (No language has this property.)
  - The set of all languages. (Every language has this property.)

  Note: There is a difference between the following properties:
  1. Ø This means that the language has no strings. It is a trivial property.
  2. {Ø} This means that the language contains nothing. It is a nontrivial property.

  Ø is a language containing no string. {Ø} is a language containing exactly one string, the empty string, which has length 0.
- **Proof:**
  Let P be a nontrivial property of recognizable languages.
  It suffices to prove that $U \leq_m L_P$.
  Either $\{Ø\} \in P$ or $\{Ø\} \notin P$.

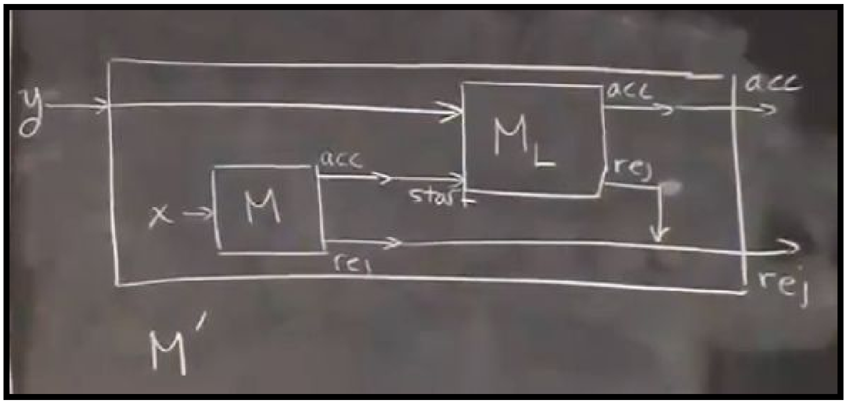
  <u>Case 1: $\{Ø\} \notin P$</u>
  Since P is nontrivial, some recognizable language, L, is in P.
  I.e. $L \in P$
  $L \neq Ø$ because L belongs to P and Ø doesn't belong to P.
  Let M_L be a TM such that $L(M_L) = L$.
  I.e. M_L recognizes L.
  We want a mapping reduction from U to L_P.

  Given ⟨M,x⟩ to U, we want to construct a machine M' to L_P such that if M accepts x then L(M') has property P and if M does not accept x, then L(M') does not have property P.

  We will use L as an example of a language that has property P and we will use the empty language, L = Ø, as a language that doesn't have property P.

  f on input ⟨M,x⟩ does the following:
  1. Define M'. M' on input y does the following:
     a. Run M on x.
     b. If M accepts, then
     c.   Run M_L on y
     d.   If M_L accepts, then accept.
     e.   Else, reject
     f. Else, reject
  2. Return ⟨M'⟩

This is a diagram of the proof:



If M accepts x, then L(M') = L. This means that ⟨M'⟩ ∈ L_P.
If M does not accept x, then L(M') = Ø. No string is accepted. This means that ⟨M'⟩ ∉ L_P.

Case 2: {Ø} ∈ P
Consider ¬P, the complement of property P.
¬P = {L | L ∉ P}
I.e. ¬P is the set of languages that do not have property P.
Since {Ø} ∈ P, it follows that {Ø} ∉ ¬P.
Now, we can apply case 1 to ¬P.
By case 1, $L_{\neg P}$ is undecidable.
By 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.", we know that if language L is undecidable, then its complement, ¬L, is also undecidable.
Hence, we know that ¬$L_{\neg P}$ is also undecidable.
¬$L_{\neg P}$ is the set of languages that have the property P.
I.e. ¬$L_{\neg P}$ is simply L_P.
So L_P is undecidable.