Decidability and Undecidability Proofs

Sections 21.1 – 21.3
# Undecidable Problems
(Languages That Aren’t In D)

<table>
<thead>
<tr>
<th>The Problem View</th>
<th>The Language View</th>
</tr>
</thead>
<tbody>
<tr>
<td>Does TM $M$ halt on $w$?</td>
<td>$H = {&lt;M, w&gt;: M \text{ halts on } w}$</td>
</tr>
<tr>
<td>Does TM $M$ not halt on $w$?</td>
<td>$\neg H = {&lt;M, w&gt;: M \text{ does not halt on } w}$</td>
</tr>
<tr>
<td>Does TM $M$ halt on the empty tape?</td>
<td>$H_\epsilon = {&lt;M&gt;: M \text{ halts on } \epsilon}$</td>
</tr>
<tr>
<td>Is there any string on which TM $M$ halts?</td>
<td>$H_{\text{ANY}} = {&lt;M&gt;: \text{there exists at least one string on which TM } M \text{ halts}}$</td>
</tr>
<tr>
<td>Does TM $M$ accept all strings?</td>
<td>$A_{\text{ALL}} = {&lt;M&gt;: L(M) = \Sigma^*}$</td>
</tr>
<tr>
<td>Do TMs $M_a$ and $M_b$ accept the same languages?</td>
<td>$\text{EqTM} = {&lt;M_a, M_b&gt;: L(M_a) = L(M_b)}$</td>
</tr>
<tr>
<td>Is the language that TM $M$ accepts regular?</td>
<td>$\text{TMreg} = {&lt;M&gt;: L(M) \text{ is regular}}$</td>
</tr>
</tbody>
</table>
Reduction is Ubiquitous

- Calling Jen
  - Call Jen
  - Get hold of Jim
- Crisis detection via pizza orders
  - Show that crisis exists
    - Show spike in pizza orders
- Fixing dinner
  - Fix dinner
    - Fix entrée
    - Fix salad
    - Fix dessert
More Examples of Reduction

- Theorem proving

Suppose that we want to establish \( Q(A) \) and that we have, as a theorem:

\[
\forall x \ (R(x) \land S(x) \land T(x) \rightarrow Q(x)).
\]

\[
\begin{array}{c}
\text{Q(A)} \\
\text{R(A)} \quad \text{S(A)} \quad \text{T(A)}
\end{array}
\]
Nim

At each turn, a player chooses one pile and removes some sticks.

The player who takes the last stick wins.

Problem: Is there a move that guarantees a win for the current player?
Nim

- Obvious approach: search the space of possible moves.

- Reduction to an XOR computation problem:

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>1</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>101</td>
<td>1</td>
<td>01</td>
<td></td>
</tr>
<tr>
<td>010</td>
<td>0</td>
<td>11</td>
<td></td>
</tr>
<tr>
<td>011</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- XOR them together:
  - 0+ means state is losing for current player
  - otherwise current player can win by making a move that makes the XOR 0.
More Examples of Reduction

- Computing a function

\[ \text{multiply}(x, y) = \]
  1. \( answer := 0. \)
  2. For \( i := 1 \) to \( |y| \) do:
     \[ answer = answer + x. \]
  3. Return \( answer. \)
Using Reduction for Undecidability

**Theorem:** There exists no general procedure to solve the following problem:

Given an angle $A$, divide $A$ into sixths using only a straightedge and a compass.

**Proof:** Suppose that there were such a procedure, which we’ll call *sixth*. Then we could trisect an arbitrary angle:

\[ \text{trisect}(a: \text{angle}) = \]

But we know that *trisect* does not exist.

So:
Using Reduction for Undecidability

**Theorem:** There exists no general procedure to solve the following problem:

Given an angle $A$, divide $A$ into sixths using only a straightedge and a compass.

**Proof:** Suppose that there were such a procedure, which we’ll call *sixth*. Then we could trisect an arbitrary angle:

$\text{trisect}(a: \text{angle}) =$

1. Divide $a$ into six equal parts by invoking $\text{sixth}(a)$.
2. Ignore every other line, thus dividing $a$ into thirds.

But we know that $\text{trisect}$ does not exist. So:
Using Reduction for Undecidability

A reduction $R$ from $L_1$ to $L_2$ is one or more Turing machines such that:

If there exists a Turing machine $Oracle$ that decides (or semidecides) $L_2$, then the Turing machines in $R$ can be composed with $Oracle$ to build a deciding (or a semideciding) Turing machine for $L_1$.

$P \leq P'$ means that $P$ is reducible to $P'$. 
Using Reduction for Undecidability

$(R \text{ is a reduction from } L_1 \text{ to } L_2) \land (L_2 \text{ is in } D) \rightarrow (L_1 \text{ is in } D)$

If $(L_1 \text{ is in } D)$ is false, then at least one of the two antecedents of that implication must be false. So:

If $(R \text{ is a reduction from } L_1 \text{ to } L_2)$ is true, then $(L_2 \text{ is in } D)$ must be false.
Using Reduction for Undecidability

Showing that $L_2$ is not in $D$:

$L_1$ (known not to be in $D$) $\quad L_1$ in $D$ $\quad$ But $L_1$ not in $D$

$R$

$L_2$ (a new language whose decidability we are trying to determine) $\quad$ if $L_2$ in $D$ $\quad$ So $L_2$ not in $D$
To Use Reduction for Undecidability

1. Choose a language $L_1$:
   ● that is already known not to be in D, and
   ● that can be reduced to $L_2$.

2. Define the reduction $R$.

3. Describe the composition $C$ of $R$ with Oracle.

4. Show that $C$ does correctly decide $L_1$ iff Oracle exists. We do this by showing:
   ● $R$ can be implemented by Turing machines,
   ● $C$ is correct:
     ● If $x \in L_1$, then $C(x)$ accepts, and
     ● If $x \notin L_1$, then $C(x)$ rejects.
Mapping Reductions

$L_1$ is \textit{mapping reducible} to $L_2$ ($L_1 \leq_M L_2$) iff there exists some computable function $f$ such that:

$$\forall x \in \Sigma^* \ (x \in L_1 \iff f(x) \in L_2).$$

To decide whether $x$ is in $L_1$, we transform it, using $f$, into a new object and ask whether that object is in $L_2$.

Example:

$$\text{DecideNIM}(x) = \text{XOR-solve}(\text{transform}(x))$$
$H_\varepsilon = \{<M>: \text{TM } M \text{ halts on } \varepsilon\}$
\( H_\varepsilon = \{<M> : \text{TM } M \text{ halts on } \varepsilon \} \)

\( H_\varepsilon \) is in SD. \( T \) semidecides it:

\[
T(<M>) = \\
1. \text{Run } M \text{ on } \varepsilon. \\
2. \text{Accept.}
\]

\( T \) accepts \(<M>\) iff \( M \) halts on \( \varepsilon \), so \( T \) semidecides \( H_\varepsilon \).
Theorem: $H_\varepsilon = \{<M> : \text{TM } M \text{ halts on } \varepsilon\}$ is not in $D$.

Proof: by reduction from $H$:

$$ H = \{<M, w> : \text{TM } M \text{ halts on input string } w\} $$

$$ R \downarrow $$

(Oracle) $H_\varepsilon = \{<M> : \text{TM } M \text{ halts on } \varepsilon\}$

$R$ is a mapping reduction from $H$ to $H_\varepsilon$:

$R(<M, w>) =$

1. Construct $<M\#>$, where $M\#(x)$ operates as follows:
   
   1.1. Erase the tape.
   1.2. Write $w$ on the tape.
   1.3. Run $M$ on $w$.

2. Return $<M\#>$. 
Proof, Continued

\[ R(<M, w>) = \]

1. Construct \(<M#>\), where \(M#(x)\) operates as follows:
   1.1. Erase the tape.
   1.2. Write \(w\) on the tape.
   1.3. Run \(M\) on \(w\).

2. Return \(<M#>\).

If Oracle exists, \(C = Oracle(R(<M, w>))\) decides \(H\):

- \(C\) is correct: \(M#\) ignores its own input. It halts on everything or nothing. So:
  - \(<M, w> \in H: M\) halts on \(w\), so \(M#\) halts on everything. In particular, it halts on \(\varepsilon\). Oracle accepts.
  - \(<M, w> \notin H: M\) does not halt on \(w\), so \(M#\) halts on nothing and thus not on \(\varepsilon\). Oracle rejects.
A Block Diagram of C
$R$ Can Be Implemented as a Turing Machine

$R$ must construct $<M\#>$ from $<M, w>$. Suppose $w = \text{aba}$.

$M\#$ will be:

So the procedure for constructing $M\#$ is:

1. Write:

2. For each character $x$ in $w$ do:
   2.1. Write $x$.
   2.2. If $x$ is not the last character in $w$, write R.

3. Write $L\square M$. 
Conclusion

$R$ can be implemented as a Turing machine.

$C$ is correct.

So, if $Oracle$ exists:

$$C = Oracle(R(<M, w>))$$ decides $H$.

But no machine to decide $H$ can exist.

So neither does $Oracle$. 
This Result is Somewhat Surprising

If we could decide whether $M$ halts on the specific string $\varepsilon$, we could solve the more general problem of deciding whether $M$ halts on an arbitrary input.

Clearly, the other way around is true: If we could solve $H$ we could decide whether $M$ halts on any one particular string.

But doing a reduction in that direction would tell us nothing about whether $H_\varepsilon$ was decidable.

The significant thing that we just saw in this proof is that there also exists a reduction in the direction that does tell us that $H_\varepsilon$ is not decidable.
How Many Languages Are We Dealing With?

\[ H = \{ \langle M, w \rangle : \text{TM } M \text{ halts on input string } w \} \]

\[ R \]

(Oracle) \[ H_\varepsilon = \{ \langle M \rangle : \text{TM } M \text{ halts on } \varepsilon \} \]

H contains strings of the form:
\[(q_00, a_00, q_01, a_10, \leftarrow), (q_00, a_00, q_01, a_10, \rightarrow), \ldots, \text{aaa} \]

\[ H_\varepsilon \text{ contains strings of the form:} \]
\[(q_00, a_00, q_01, a_10, \leftarrow), (q_00, a_00, q_01, a_10, \rightarrow), \ldots \]

The language on which some \( M \) halts contains strings of some arbitrary form, for example,

\[(\text{letting } \Sigma = \{a, b\}): \text{ aaaba} \]
How Many Machines Are We Dealing With?

\[ H = \{<M, w> : \text{TM } M \text{ halts on input string } w\} \]

\[ R \]

\[ (\text{?Oracle}) \quad \quad H_\varepsilon = \{<M> : \text{TM } M \text{ halts on } \varepsilon\} \]

\( R \) is a reduction from \( H \) to \( H_\varepsilon \):

\[ R(<M, w>) = \]

1. Construct \( <M#> \), where \( M#(x) \) operates as follows:
   1.1. Erase the tape.
   1.2. Write \( w \) on the tape.
   1.3. Run \( M \) on \( w \).

2. Return \( <M#> \).

- \( \text{Oracle} \) (the hypothesized machine to decide \( H_\varepsilon \)).
- \( R \) (the machine that builds \( M# \). Actually exists).
- \( C \) (the composition of \( R \) with \( \text{Oracle} \)).
- \( M# \) (the machine we will pass as input to \( \text{Oracle} \)). Note that we never run it.
- \( M \) (the machine whose membership in \( H \) we are interested in determining; thus also an input to \( R \)).
Important Elements in a Reduction Proof

• A clear declaration of the reduction “from” and “to” languages.
• A clear description of $R$.
• If $R$ is doing anything nontrivial, argue that it can be implemented as a TM.
• Note that machine diagrams are not necessary or even sufficient in these proofs. Use them as thought devices, where needed.
• Run through the logic that demonstrates how the “from” language is being decided by the composition of $R$ and Oracle. You must do both accepting and rejecting cases.
• Declare that the reduction proves that your “to” language is not in D.
Another Way to View the Reduction

// let \( L = \{<M> \mid M \text{ is a TM that halts on epsilon}\} \)
// if \( L \) is recursive, let this function decide \( L \):

```cpp
bool HaltsOnEpsilon(TM M); // defined in magic.h
```

// \( \text{HaltsOn} \) decides \( H \) using \( \text{HaltsOnEpsilon} \)
// .: \( \text{HaltsOn} \) reduces to \( \text{HaltsOnEpsilon} \) as such:

```cpp
bool HaltsOn(TM M, string w)
{
  // a nested TM
  void Wrapper(string idontcare) {
    M(w);
  }

  return HaltsOnEpsilon(Wrapper);
}
```
The Most Common Mistake: Doing the Reduction Backwards

The right way to use reduction to show that $L_2$ is not in D:

1. Given that $L_1$ is not in D,
2. Reduce $L_1$ to $L_2$, i.e., show how to solve $L_1$ (the known one) in terms of $L_2$ (the unknown one)

Doing it wrong by reducing $L_2$ (the unknown one to $L_1$):

If there exists a machine $M_1$ that solves $H$, then we could build a machine that solves $L_2$ as follows:

1. Return $(M_1(<M, \varepsilon>))$.

This proves nothing. It’s an argument of the form:

If False then …
\( H_{\text{ANY}} = \{ <M> : \text{there exists at least one string on which TM } M \text{ halts} \} \)

**Theorem:** \( H_{\text{ANY}} \) is in SD.

**Proof:** by exhibiting a TM \( T \) that semidecides it.

What about simply trying all the strings in \( \Sigma^* \) one at a time until one halts?
$H_{\text{ANY}}$ is in SD

$T(<M>) =$

1. Use dovetailing to try $M$ on all of the elements of $\Sigma^*$:

\[
\begin{align*}
\varepsilon & \ [1] \\
\varepsilon & \ [2] \ a \ [1] \\
\varepsilon & \ [3] \ a \ [2] \ b \ [1] \\
\varepsilon & \ [4] \ a \ [3] \ b \ [2] \ aa \ [1] \\
\varepsilon & \ [5] \ a \ [4] \ b \ [3] \ aa \ [2] \ ab \ [1]
\end{align*}
\]

2. If any instance of $M$ halts, halt and accept.

$T$ will accept iff $M$ halts on at least one string. So $T$ semidecides $H_{\text{ANY}}$. 
H<sub>ANY</sub> is not in D

\[ H = \{ <M, w> : \text{TM } M \text{ halts on input string } w \} \]

\[ R \]

(Oracle) \quad H_{ANY} = \{ <M> : \text{there exists at least one string on which TM } M \text{ halts} \}

\[ R(<M, w>) = \]

1. Construct \(<M#>, \text{where } M#(x) \text{ operates as follows:} \]
   1.1. Examine \(x\).
   1.2. If \(x = w\), run \(M\) on \(w\), else loop.
2. Return \(<M#>\).

If Oracle exists, then \(C = Oracle(R(<M, w>))\) decides \(H\):
- \(R\) can be implemented as a Turing machine.
- \(C\) is correct: The only string on which \(M#\) can halt is \(w\). So:
  - \(<M, w> \in H: M \text{ halts on } w\). So \(M#\) halts on \(w\). There exists at least one string on which \(M#\) halts. Oracle accepts.
  - \(<M, w> \notin H: M \text{ does not halt on } w\), so neither does \(M#\). So there exists no string on which \(M#\) halts. Oracle rejects.

But no machine to decide \(H\) can exist, so neither does Oracle.
(Another $R$ That Works)

**Proof:** We show that $H_{\text{ANY}}$ is not in $D$ by reduction from $H$:

$$H = \{<M, w> : \text{TM } M \text{ halts on input string } w\}$$

$$R(\ ?\text{Oracle})$$

$$H_{\text{ANY}} = \{<M> : \text{there exists at least one string on which TM } M \text{ halts}\}$$

$R(<M, w>) =$

1. Construct the description $<M#>$, where $M#(x)$ operates as follows:
   1.1. Erase the tape.
   1.2. Write $w$ on the tape.
   1.3. Run $M$ on $w$.
2. Return $<M#>$.

If Oracle exists, then $C = \text{Oracle}(R(<M, w>))$ decides $H$:

- $C$ is correct: $M#$ ignores its own input. It halts on everything or nothing. So:
  - $<M, w> \in H$: $M$ halts on $w$, so $M#$ halts on everything. So it halts on at least one string. Oracle accepts.
  - $<M, w> \notin H$: $M$ does not halt on $w$, so $M#$ halts on nothing. So it does not halt on at least one string. Oracle rejects.

But no machine to decide $H$ can exist, so neither does Oracle.
The Steps in a Reduction Proof

1. ⭐ Choose an undecidable language to reduce from.

2. ⭐ Define the reduction $R$.

3. Show that $C$ (the composition of $R$ with $Oracle$) is correct.

⭐ indicates where we make choices.
\[ H_{\text{ALL}} = \{ <M> : \text{TM } M \text{ halts on all inputs} \} \]
We show that $H_{\text{ALL}}$ is not in D by reduction from $H_\varepsilon$.

$$H_\varepsilon = \{ <M> : \text{TM } M \text{ halts on } \varepsilon \}$$

$R(?) = (\text{Oracle})$

$$H_{\text{ALL}} = \{ <M> : \text{TM } M \text{ halts on all inputs } \}$$

$R(<M>) =$

1. Construct the description $<M#>$, where $M#(x)$ operates as follows:
   1.1. Erase the tape.
   1.2. Run $M$.
2. Return $<M#>$.

If Oracle exists, then $C = \text{Oracle}(R(<M>))$ decides $H_\varepsilon$:

- $R$ can be implemented as a Turing machine.
- $C$ is correct: $M#$ halts on everything or nothing, depending on whether $M$ halts on $\varepsilon$. So:
  - $<M> \in H_\varepsilon$: $M$ halts on $\varepsilon$, so $M#$ halts on all inputs. Oracle accepts.
  - $<M> \notin H_\varepsilon$: $M$ does not halt on $\varepsilon$, so $M#$ halts on nothing. Oracle rejects.

But no machine to decide $H_\varepsilon$ can exist, so neither does Oracle.
We next define a new language:

\[ A = \{ <M, w> : M \text{ accepts } w \}. \]

Note that \( A \) is different from \( H \) since it is possible that \( M \) halts but does not accept. An alternative definition of \( A \) is:

\[ A = \{ <M, w> : w \in L(M) \}. \]
\[ A = \{ <M, w> : w \in L(M) \} \]

We show that A is not in D by reduction from H.

\[ H = \{ <M, w> : \text{TM } M \text{ halts on input string } w \} \]

\[ R(\text{Oracle}) \quad A = \{ <M, w> : w \in L(M) \} \]

\[ R(<M, w>) = \]

1. Construct the description \(<M\#>\), where \(M\#(x)\) operates as follows:
   1.1. Erase the tape.
   1.2. Write \(w\) on the tape.
   1.3. Run \(M\) on \(w\).
   1.4. **Accept**
2. Return \(<M\#, w>\).

If \(Oracle\) exists, then \(C = Oracle(R(<M, w>))\) decides H:

- \(R\) can be implemented as a Turing machine.
- \(C\) is correct: \(M\#\) accepts everything or nothing. So:
  - \(<M, w> \in H: M\) halts on \(w\), so \(M\#\) accepts everything. In particular, it accepts \(w\). \(Oracle\) accepts.
  - \(<M, w> \notin H: M\) does not halt on \(w\). \(M\#\) gets stuck in step 1.3 and so accepts nothing. \(Oracle\) rejects.

But no machine to decide H can exist, so neither does \(Oracle\).
**Theorem:** $A_{\varepsilon} = \{<M> : \text{TM } M \text{ accepts } \varepsilon\}$ is not in D.

**Proof:** Analogous to that for $H_{\varepsilon}$.

**Theorem:**

$A_{\text{ANY}} = \{<M> : \text{TM } M \text{ accepts at least one string}\}$

is not in D.

**Proof:** Analogous to that for $H_{\text{ANY}}$.

**Theorem:** $A_{\text{ALL}} = \{<M> : L(M) = \Sigma^*\}$ is not in D.

**Proof:** Analogous to that for $H_{\text{ALL}}$. 

**A\_\varepsilon, A\_\text{ANY}, and A\_\text{ALL}**
### Are Security Properties Decidable?

<table>
<thead>
<tr>
<th></th>
<th>process₁</th>
<th>process₂</th>
<th>process₃</th>
<th>process₄</th>
<th>file₁</th>
<th>file₂</th>
</tr>
</thead>
<tbody>
<tr>
<td>process₁</td>
<td>execute</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>process₂</td>
<td></td>
<td>execute</td>
<td>execute</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>process₃</td>
<td></td>
<td></td>
<td>execute</td>
<td></td>
<td></td>
<td>own</td>
</tr>
<tr>
<td>process₄</td>
<td></td>
<td></td>
<td></td>
<td>execute</td>
<td></td>
<td>read</td>
</tr>
</tbody>
</table>

Suppose we could show that rights don’t “leak”.

Then we could encode configurations of TMs in this framework:

```
q      a   b   b   q  
```

Suppose we could show that rights don’t “leak”.

Then we could encode configurations of TMs in this framework:

```
q a b b q ...
```

```
<table>
<thead>
<tr>
<th></th>
<th>s₁</th>
<th>s₂</th>
<th>s₃</th>
<th>s₄</th>
</tr>
</thead>
<tbody>
<tr>
<td>s₁</td>
<td>q</td>
<td>own</td>
<td></td>
<td></td>
</tr>
<tr>
<td>s₂</td>
<td>a</td>
<td>own</td>
<td></td>
<td></td>
</tr>
<tr>
<td>s₃</td>
<td>b, q₅</td>
<td>own</td>
<td></td>
<td></td>
</tr>
<tr>
<td>s₄</td>
<td></td>
<td></td>
<td>b, end</td>
<td></td>
</tr>
</tbody>
</table>
```
Are Security Properties Decidable?

Now we can decide $H_\varepsilon$ by reducing it to the leakage problem:

Modify $M$ so that it has a single halting state $q_f$.

Start configuration:

<table>
<thead>
<tr>
<th></th>
<th>$s_1$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$s_1$</td>
<td>$\square, q_1, \text{end}$</td>
</tr>
</tbody>
</table>

The question to ask: Does $q_f$ ever show up in any cell?
EqTMs=$\{<M_a, M_b>: L(M_a)=L(M_b)\}$

Oracle for EqTMs
EqTMs = \{<M_a, M_b>: L(M_a) = L(M_b)\}

A_{\text{ANY}} = \{<M>: \text{there exists at least one string on which TM } M \text{ halts}\}

R(\langle M \rangle) =
1. Construct the description of \(M\#(x)\):
   1.1. Accept.
   2. Return \(\langle M, M\# \rangle\).

If Oracle exists, then \(C = \text{Oracle}(R(\langle M \rangle))\) decides \(A_{\text{ANY}}\):
   • \(C\) is correct: \(M\#\) accepts everything. So:
     • \(\langle M \rangle \in A_{\text{ANY}}: L(M) =? L(M\#). \text{ Oracle?}\)
     • \(\langle M \rangle \notin A_{\text{ANY}}: L(M) \neq L(M\#). \text{ Oracle rejects.}\)
EqTMs = \{<M_a, M_b>: L(M_a) = L(M_b)\}

A_{\text{ALL}} = \{<M>: L(M) = \Sigma^*\}

R(\langle M \rangle) =
1. Construct the description of \( M\#(x) \):
   1.1. Accept.
2. Return \( \langle M, M\# \rangle \).

If Oracle exists, then \( C = \text{Oracle}(R(\langle M \rangle)) \) decides \( A_{\text{ALL}} \):
   • \( C \) is correct: \( M\# \) accepts everything. So if \( L(M) = L(M\#) \), \( M \) must also accept everything. So:
     • \( \langle M \rangle \in A_{\text{ALL}}: L(M) = L(M\#) \). Oracle accepts.
     • \( \langle M \rangle \notin A_{\text{ALL}}: L(M) \neq L(M\#) \). Oracle rejects.

But no machine to decide \( A_{\text{ALL}} \) can exist, so neither does Oracle.
A Practical Consequence

Consider the problem of virus detection. Suppose that a new virus $V$ is discovered and its code is $<V>$. 

- Is it sufficient for antivirus software to check solely for occurrences of $<V>$?
- Is it possible for it to check for equivalence to $V$?
Sometimes Mapping Reducibility Isn’t Right

Recall that a mapping reduction from $L_1$ to $L_2$ is a computable function $f$ where:

$$\forall x \in \Sigma^* \ (x \in L_1 \iff f(x) \in L_2).$$

When we use a mapping reduction, we return:

$$Oracle(f(x))$$

Sometimes we need a more general ability to use $Oracle$ as a subroutine and then to do other computations after it returns.
{<M> : M accepts no even length strings}

H = {< M, w> : TM M halts on input string w}

R

(?Oracle) L_2 = {<M> : M accepts no even length strings}

R(<M, w>) =
1. Construct the description <M#>, where M#(x) operates as follows:
   1.1. Erase the tape.
   1.2. Write w on the tape.
   1.3. Run M on w.
   1.4. Accept.
2. Return <M#>.

If Oracle exists, then C = Oracle(R(<M, w>)) decides H:
- C is correct: M# ignores its own input. It accepts everything or nothing, depending on whether it makes it to step 1.4. So:
  - <M, w> ∈ H: M halts on w. Oracle:
  - <M, w> ∉ H: M does not halt on w. Oracle:

Problem:
\{<M> : M accepts no even length strings\}

\[ H = \{<M, w> : \text{TM } M \text{ halts on input string } w\} \]

\[ \downarrow R \]

(?Oracle) \[ L_2 = \{<M> : M \text{ accepts no even length strings}\} \]

\[ R(<M, w>) = \]
1. Construct the description \(<M\#>\), where \(M\#(x)\) operates as follows:
   1.1. Erase the tape.
   1.2. Write \(w\) on the tape.
   1.3. Run \(M\) on \(w\).
   1.4. Accept.
2. Return \(<M\#>\).

If Oracle exists, then \(C = \neg \text{Oracle}(R(<M, w>))\) decides \(H\):

- \(R\) and \(\neg\) can be implemented as Turing machines.
- \(C\) is correct:
  - \(<M, w> \in H: M \text{ halts on } w. M\# \text{ accepts everything, including some even length strings. Oracle rejects so C accepts.}\)
  - \(<M, w> \notin H: M \text{ does not halt on } w. M\# \text{ gets stuck. So it accepts nothing, so no even length strings. Oracle accepts. So C rejects}.\)

But no machine to decide \(H\) can exist, so neither does Oracle.
Are All Questions about TMs Undecidable?

Let $L = \{<M> : \text{TM } M \text{ contains an even number of states}\}$
Are All Questions about TMs Undecidable?

Let $L = \{<M, w> : M \text{ halts on } w \text{ within 3 steps}\}$. 
Another One

Let $L_q = \{<M, q> : \text{there is some configuration}
(p, uav) \text{ of } M, \text{ with } p \neq q,\n$ that yields a configuration whose state is $q\}.

Is $L_q$ decidable?