Decidability and Undecidability Proofs

Sections 21.4 - 21.7
Is There a Pattern?

- Does $L$ contain some particular string $w$?
- Does $L$ contain $\varepsilon$?
- Does $L$ contain any strings at all?
- Does $L$ contain all strings over some alphabet $\Sigma$?

- $A = \{<M, w> : \text{TM } M \text{ accepts } w\}$.
- $A_\varepsilon = \{<M> : \text{TM } M \text{ accepts } \varepsilon\}$.
- $A_{\text{ANY}} = \{<M> : \text{there exists at least one string that TM } M \text{ accepts}\}$.
- $A_{\text{ALL}} = \{<M> : \text{TM } M \text{ accepts all inputs}\}$.
Rice’s Theorem

No nontrivial property of the SD languages is decidable.

or

Any language that can be described as:

\{<M>: P(L(M)) = True\}

for any nontrivial property \(P\), is not in D.

A nontrivial property is one that is not simply:

• True for all languages, or
• False for all languages.
Applying Rice’s Theorem

To use Rice’s Theorem to show that a language $L$ is not in $D$ we must:

- Specify property $P$.
- Show that the domain of $P$ is the SD languages.
- Show that $P$ is nontrivial:
  - $P$ is true of at least one language
  - $P$ is false of at least one language
Applying Rice’s Theorem

1. \{<M> : L(M) contains only even length strings}\.
2. \{<M> : L(M) contains an odd number of strings}\.
3. \{<M> : L(M) contains all strings that start with a\}.
4. \{<M> : L(M) is infinite\}.
5. \{<M> : L(M) is regular\}.
6. \{<M> : M contains an even number of states\}.
7. \{<M> : M has an odd number of symbols in its tape alphabet\}.
8. \{<M> : M accepts \(\varepsilon\) within 100 steps\}.
9. \{<M> : M accepts \(\varepsilon\)\}.
10. \{<M_a, M_b> : L(M_a) = L(M_b)\}.
Proof of Rice’s Theorem

**Proof:** Let $P$ be any nontrivial property of the SD languages.

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

$$R \downarrow$$

(Oracle) $L_2 = \{<M> : P(L(M)) = T\}$

Either $P(\emptyset) = T$ or $P(\emptyset) = F$. Assume it is $F$ (a matching proof exists if it is $T$).

Since $P$ is nontrivial, there is some SD language $L_T$ such that $P(L_T)$ is $T$. Let $K$ be some Turing machine that semidecides $L_T$. 

Rice’s Theorem

Proof of Rice’s Theorem

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

1. Construct \(<M#>\), so \(M#(x)\) operates as follows:
   1.1. Copy its input \(x\) to another track for later.
   1.2. Erase the tape.
   1.3. Write \(w\) on the tape.
   1.4. Run \(M\) on \(w\).
   1.5. Put \(x\) back on the tape and run \(K\) on \(x\).

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

Recall: \(K\) decides \(L_T\) and we know that \(P(L_T)\) is True.
Proof, Continued

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

**Proof:**
- \( R \) can be implemented as a Turing machine.
- \( C \) is correct:

But no machine to decide H can exist, so neither does Oracle.
Proof that $R$ Can Be Implemented

1. Construct $<M#>$, so $M#(x)$ operates as follows:
   1.1. Copy its input $x$ to another track for later.
   1.2. Erase the tape.
   1.3. Write $w$ on the tape.
   1.4. Run $M$ on $w$.
   1.5. Put $x$ back on the tape and run $K$ on $x$.
2. Return $<M#>$.
Proof that \( C \) is Correct

- \(<M, w> \in H: M \) halts on \( w \). \( M\# \) makes it to 1.5. So it is equivalent to \( K \).
  \[ L(M\#) = L(K) \text{ and } P(L(M\#)) = P(L(K)). \]
  \( P(L(K)) \) is \( T \), so \( P(L(M\#)) \) is \( T \).
  Oracle decides \( P \). Oracle accepts.

- \(<M, w> \notin H: M \) does not halt on \( w \). \( M\# \) gets stuck in 1.4. So it accepts nothing.
  \[ L(M\#) = \emptyset. \quad P(\emptyset) = F. \]
  Oracle decides \( P \). Oracle rejects.

***********

1. Construct \(<M\#>\), so \( M\#(x) \) operates as follows:
   1.1. Copy its input \( x \) to another track for later.
   1.2. Erase the tape.
   1.3. Write \( w \) on the tape.
   1.4. Run \( M \) on \( w \).
   1.5 Put \( x \) back on the tape and run \( K \) on \( x \).
2. Return \(<M\#>\).
Given a TM $M$, is $L(M)$ Regular?

The problem: Is $L(M)$ regular?

As a language: Is $\{<M> : L(M) \text{ is regular}\}$ in $D$?

No, by Rice’s Theorem:

- $P = True$ if $L$ is regular and $False$ otherwise.
- The domain of $P$ is the set of SD languages since it is the set of languages accepted by some TM.
- $P$ is nontrivial:
  - $P(a^*) = True$.
  - $P(A^nB^n) = False$. 
Given a Turing Machine $M$, is $L(M)$ Regular?

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

$$R$$

(Oracle) $$L_2 = \{<M> : L(M) \text{ is regular}\}$$

$$R(<M, w>) =$$

1. Construct the description of $M#(x)$:
   1.1. Erase tape.
   1.2. Write $w$ on the tape.
   1.3. Run $M$ on $w$.

2. Return $<M#>$.

Is this correct?
Given a Turing Machine $M$, is $L(M)$ Regular?

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

\[
R
\]

(Oracle) \hspace{1cm} L_2 = \{<M> : L(M) \text{ is regular}\}

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

1. Construct $M\#(x)$:
   
   1.1. Erase tape.
   
   1.2. Write $w$ on the tape.
   
   1.3. Run $M$ on $w$.
   
   1.4. Accept

2. Return $<M\#>$.

Is this correct?
Given a Turing Machine $M$, is $L(M)$ Regular?

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

$R$ is an oracle that computes:

$R(<M, w>) =$

1. Construct $M#(x)$:
   1.1. Copy its input $x$ to another track for later.
   1.2. Erase the tape.
   1.3. Write $w$ on the tape.
   1.4. Run $M$ on $w$.
   1.5. Put $x$ back on the tape.
   1.6. If $x \in A^nB^n$ then accept, else reject.

2. Return $<M#>$.

Problem:
But We Can Flip

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

1. Construct the description \(<M\#>\), where \(M\#(x)\) operates as follows:
   1.1. Save \(x\) for later.
   1.2. Erase the tape.
   1.3. Write \(w\) on the tape.
   1.4. Run \(M\) on \(w\).
   1.5. Put \(x\) back on the tape.
   1.6. If \(x \in A^nB^n\) then accept, else reject.
2. Return \(<M\#>\).

If Oracle decides \(L_2\), then \(C = \neg Oracle(R(<M, w>))\) decides \(H\):
   ● \(<M, w> \in H\): \(M\#\) makes it to step 1.5. Then it accepts \(x\) iff \(x \in A^nB^n\). So \(M\#\) accepts \(A^nB^n\), which is not regular.
      Oracle rejects. \(C\) accepts.
   ● \(<M, w> \notin H\): \(M\) does not halt on \(w\). \(M\#\) gets stuck in step 1.4.
      It accepts nothing. \(L(M\#) = \emptyset\), which is regular.
      Oracle accepts. \(C\) rejects.

But no machine to decide \(H\) can exist, so neither does \(Oracle\).
Or, Doing it Without Flipping

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

1. Construct the description \(<M\#>\), where \(M\#(x)\) operates as follows:
   1.1. If \(x \in A^nB^n\) then accept, else:
   1.2. Erase the tape.
   1.3. Write \(w\) on the tape.
   1.4. Run \(M\) on \(w\).
   1.5. Accept

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

If \(Oracle\) exists, \(C = Oracle(R(<M, w>))\) decides \(H\):
- \(C\) is correct: \(M\#\) immediately accepts all strings \(A^nB^n\):
  - \(<M, w> \in H: M\#\) accepts everything else in step 1.5. So \(L(M\#) = \Sigma^*\), which is regular. \(Oracle\) accepts.

  - \(<M, w> \notin H: M\#\) gets stuck in step 1.4, so it accepts nothing else. \(L(M\#) = A^nB^n\), which is not regular. \(Oracle\) rejects.

But no machine to decide \(H\) can exist, so neither does \(Oracle\).
Any Nonregular Language Will Work

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

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

If Oracle exists, \(C = Oracle(R(<M, w>))\) decides H:
- C is correct: \(M\#\) immediately accepts all strings WW:
  - \(<M, w> \in H: M\#\) accepts everything else in step 1.5. So \(L(M\#) = \Sigma^*\), which is regular. Oracle accepts.
  - \(<M, w> \notin H: M\#\) gets stuck in step 1.4, so it accepts nothing else. \(L(M\#) = WW\), which is not regular. Oracle rejects.

But no machine to decide H can exist, so neither does Oracle.
Is $L(M)$ Context-free?

How about: $L_3 = \{<M> : L(M) \text{ is context-free}\}$?
Is $L(M)$ Context-free?

How about: $L_3 = \{<M>: L(M) \text{ is context-free}\}$?

$R(<M, w>) =$
1. Construct the description $<M\#>$, where $M\#(x)$ operates as follows:
   1.1. If $x \in A^n B^n C^n$ then accept, else:
   1.2. Erase the tape.
   1.3. Write $w$ on the tape.
   1.4. Run $M$ on $w$.
   1.5. Accept
2. Return $<M\#>$.
Practical Impact of These Results

1. Does $P$, when running on $x$, halt?
2. Might $P$ get into an infinite loop on some input?
3. Does $P$, when running on $x$, ever output a 0? Or anything at all?
4. Are $P_1$ and $P_2$ equivalent?
5. Does $P$, when running on $x$, ever assign a value to $n$?
6. Does $P$ ever reach $S$ on any input (in other words, can we chop it out)?
7. Does $P$ reach $S$ on every input (in other words, can we guarantee that $S$ happens)?
   - Can the Patent Office check prior art?
   - Can the CS department buy the definitive grading program?
Turing Machine Questions Can be Reduced to Program Questions

EqPrograms =

\{<P_a, P_b> : P_a and P_b are PL programs and L(P_a) = L(P_b)\}.

We can build, in any programming language PL, \textit{SimUM}:

- that is a \textit{PL} program
- that implements the Universal TM U and so can simulate an arbitrary TM.
TM Questions and Program Questions

EqPrograms = \{<P_a, P_b> : P_a and P_b are PL programs and L(P_a) = L(P_b)\}.

**Theorem:** EqPrograms is not in D.

**Proof:** Reduction from EqTMs = \{<M_a, M_b> : L(M_a) = L(M_b)\}.

\[ R(<M_a, M_b>) = \]

1. Build \(P_1\), a PL program that, on \(w\), returns \(SimUM(M_a, w)\).
2. Build \(P_2\), a PL program that, on \(w\), returns \(SimUM(M_b, w)\).
3. Return \(<P_1, P_2>\).

If Oracle exists and decides EqPrograms, then \(C = Oracle(R(<M_a, M_b>))\) decides EqTMs. \(C\) is correct. \(L(P_1) = L(M_a)\) and \(L(P_2) = L(M_b)\). So:

- \(<M_a, M_b> \in \text{EqTMs}: L(M_a) = L(M_b). So L(P_1) = L(P_2). Oracle(<P_1, P_2>)\) accepts.
- \(<M_a, M_b> \notin \text{EqTMs}: L(M_a) \neq L(M_b). So L(P_1) \neq L(P_2). Oracle(<P_1, P_2>)\) rejects.

But no machine to decide EqTMs can exist, so neither does Oracle.
\{<M, q>: M \text{ reaches } q \text{ on some input}\}

\[H_{\text{ANY}} = \{<M>: \text{there exists some string on which TM } M \text{ halts}\}\]

\[R(\text{Oracle}) \]

\[L_2 = \{<M, q>: M \text{ reaches } q \text{ on some input}\}\]

\[R(<M>) = \]

1. Build \(<M\#>\) so that \(M\#\) is identical to \(M\) except that, if \(M\) has a transition \(((q_1, c_1), (q_2, c_2, d))\) and \(q_2\) is a halting state other than \(h\), replace that transition with:

\[(((q_1, c_1), (h, c_2, d)).\]

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

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

- \(R\) can be implemented as a Turing machine.
- \(C\) is correct: \(M\#\) will reach the halting state \(h\) iff \(M\) would reach some halting state. So:
  - \(<M> \in H_{\text{ANY}}\): There is some string on which \(M\) halts. So there is some string on which \(M\#\) reaches state \(h\). \text{Oracle} accepts.
  - \(<M> \notin H_{\text{ANY}}\): There is no string on which \(M\) halts. So there is no string on which \(M\#\) reaches state \(h\). \text{Oracle} rejects.

But no machine to decide \(H_{\text{ANY}}\) can exist, so neither does \text{Oracle}.
How many Turing machines does it take to change a light bulb?
How many Turing machines does it take to change a light bulb?

One.
How many Turing machines does it take to change a light bulb?

One.

How can you tell whether your Turing machine is the one?
How many Turing machines does it take to change a light bulb?

One.

How can you tell whether your Turing machine is the one?

You can’t.

- Tim Nodine
Non-SD Languages

There is an uncountable number of non-SD languages, but only a countably infinite number of TM’s (hence SD languages). \[\therefore\] The class of non-SD languages is much bigger than that of SD languages!
Non-SD Languages

**Intuition:** Non-SD languages usually involve either infinite search or knowing a TM will infinite loop.

Examples:

- \( \neg H = \{<M, w> : \text{TM } M \text{ does not halt on } w\} \).
- \( \{<M> : L(M) = \Sigma^*\} \).
- \( \{<M> : \text{TM } M \text{ halts on nothing}\} \).
Proving Languages are not SD

- Contradiction
- \( L \) is the complement of an SD/D Language.
- Reduction from a known non-SD language
**Theorem:** $TM_{MIN} = \{<M>: \text{Turing machine } M \text{ is minimal}\}$ is not in SD.

**Proof:** If $TM_{MIN}$ were in SD, then there would exist some Turing machine $ENUM$ that enumerates its elements. Define the following Turing machine:

$$M#(x) =$$

1. Invoke $obtainSelf$ to produce $<M#>$.
2. Run $ENUM$ until it generates the description of some Turing machine $M'$ whose description is longer than $|<M#>|$.
3. Invoke $U$ on the string $<M', x>$.

Since $TM_{MIN}$ is infinite, $ENUM$ must eventually generate a string that is longer than $|<M#>|$. So $M#$ makes it to step 3 and so is equivalent to $M'$ since it simulates $M'$. But, since $|<M#>| < |<M'||$, $M'$ cannot be minimal. Yet it was generated by $ENUM$. Contradiction.
The Compliment of $L$ is in SD/D

Suppose we want to know whether $L$ is in SD and we know:

- $\neg L$ is in SD, and
- At least one of $L$ or $\neg L$ is not in D.

Then we can conclude that $L$ is not in SD, because, if it were, it would force both itself and its complement into D, which we know cannot be true.

Example:
- $\neg H$ (since $\neg (\neg H) = H$ is in SD and not in D)
\[ \text{H}_{\neg\text{ANY}} \]

**Theorem:** \( \text{H}_{\neg\text{ANY}} = \{<M>: \text{there does not exist any string on which TM } M \text{ halts}\} \) is not in SD.

**Proof:** \( \neg \text{H}_{\neg\text{ANY}} \) is \( \text{H}_{\text{ANY}} = \{<M>: \text{there exists at least one string on which TM } M \text{ halts}\} \).

We already know:

- \( \neg \text{H}_{\neg\text{ANY}} \) is in SD.
- \( \neg \text{H}_{\neg\text{ANY}} \) is not in D.

So \( \text{H}_{\neg\text{ANY}} \) is not in SD because, if it were, then \( \text{H}_{\text{ANY}} \) would be in D but it isn’t.
Using Reduction

**Theorem:** If there is a reduction $R$ from $L_1$ to $L_2$ and $L_1$ is not SD, then $L_2$ is not SD.

So, we must:
- Choose a language $L_1$ that is known not to be in SD.
- Hypothesize the existence of a *semideciding* TM Oracle.

**Note:** $R$ may not swap accept for loop.
Using Reduction for $H_{\neg \text{ANY}}$

$H = \{<M, w>: \text{TM } M \text{ does not halt on input string } w\}$

$R(<M, w>) =$

1. Construct the description $<M\#>$ of $M\#(x)$:
   1.1. Erase the tape.
   1.2. Write $w$ on the tape.
   1.3. Run $M$ on $w$.
2. Return $<M\#>$. 
Or We Could Use Reduction for $H_{\neg \text{ANY}}$

$$R(<M, w>) =$$
1. Construct the description $<M#>$ of $M#(x)$:
   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 $C = Oracle(R(<M, w>))$ semidecides $\neg H$:
- $C$ is correct: $M#$ ignores its input. It halts on everything or nothing, depending on whether $M$ halts on $w$. So:
  - $<M, w> \in \neg H$: $M$ does not halt on $w$, so $M#$ halts on nothing. Oracle accepts.
  - $<M, w> \notin \neg H$: $M$ halts on $w$, so $M#$ halts on everything. Oracle does not accept.

But no machine to semidecide $\neg H$ can exist, so neither does Oracle.
$A_{anbn} = \{<M> : L(M) = A^nB^n\}$

$A_{anbn}$ contains strings that look like:

(q00, a00, q01, a00, →),
(q00, a01, q00, a10, →),
(q00, a10, q01, a01, ←),
(q00, a11, q01, a10, ←),
(q01, a00, q00, a01, →),
(q01, a01, q01, a10, →),
(q01, a10, q01, a11, ←),
(q01, a11, q11, a01, ←)

It does not contain strings like aaabbb.

But $A^nB^n$ does.
What’s wrong with this proof that \( A_{anbn} \) is not in SD:

\[
\neg H = \{ <M, w> : \text{TM } M \text{ does not halt on } w \} \\
R \downarrow
\]

(?Oracle) \( A_{anbn} = \{ <M> : L(M) = A^nB^n \} \)

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

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, \( C = Oracle(R(<M, w>)) \) semidecides \( \neg H \):
$A_{anbn} = \{<M> : L(M) = A^nB^n\}$ is not SD

What about: $\neg H = \{<M, w> : \text{TM } M \text{ does not halt on } w\}$

$R$?

(?Oracle) $A_{anbn} = \{<M> : L(M) = A^nB^n\}$

$R(<M, w>) =$

1. Construct the description $<M#>$, where $M#(x)$ operates as follows:
   1.1 Copy the input $x$ to another track for later.
   1.2. Erase the tape.
   1.3. Write $w$ on the tape.
   1.4. Run $M$ on $w$.
   1.5. Put $x$ back on the tape.
   1.6. If $x \in A^nB^n$ then accept, else loop.
2. Return $<M#>$.

If Oracle exists, $C = Oracle(R(<M, w>))$ semidecides $\neg H$: 
A_{anbn} = \{<M> : L(M) = A^nB^n\} is not SD

R(<M, w>) reduces \neg H to A_{anbn}:
1. Construct the description <M#>:
   1.1. If x \in A^nB^n then accept. Else:
   1.2. Erase the tape.
   1.3. Write w on the tape.
   1.4. Run M on w.
   1.5. Accept.
2. Return <M#>.

If Oracle exists, then C = Oracle(R(<M, w>)) semidecides \neg H:
M# immediately accepts all strings in A^nB^n. If M does not halt on w, those are the only strings M# accepts. If M halts on w, M# accepts everything:
   • <M, w> \in \neg H: M does not halt on w, so M# accepts strings in A^nB^n in step 1.1. Then it gets stuck in step 1.4, so it accepts nothing else. It is an A^nB^n acceptor. Oracle accepts.
   • <M, w> \notin \neg H: M halts on w, so M# accepts everything. Oracle does not accept.

But no machine to semidecide \neg H can exist, so neither does Oracle.
$H_{\text{ALL}} = \{<M> : \text{TM halts on } \Sigma^*\}$

What about: $\neg H = \{<M, w> : \text{TM } M \text{ does not halt on } w\}$

$R(<M, w>) =$

Reduction Attempt 1: $R(<M, w>) =$

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 $<M#>$.
There May Be No Easy Way to Flip

\[ \neg H = \{<M, w>: \text{TM } M \text{ does not halt on } w\} \]

Reduction Attempt 1: \( R(<M, w>) = \)

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, \(C = Oracle(R(<M, w>))\) semidecides \(\neg H\):

- \(<M, w> \in \neg H: M\) does not halt on \(w\), so \(M#\) gets stuck in step 1.3 and halts on nothing. \(Oracle\) does not accept.
- \(<M, w> \notin \neg H: M\) halts on \(w\), so \(M#\) halts on everything. \(Oracle\) accepts.
\[H_{\text{ALL}} = \{<M> : \text{TM halts on } \Sigma^*\}\]

\[R(<M, w>)\] reduces \(\neg H\) to \(H_{\text{ALL}}\):

1. Construct the description \(<M\#>\), where \(M\#(x)\) operates as follows:
   1.1. Copy the input \(x\) to another track for later.
   1.2. Erase the tape.
   1.3. Write \(w\) on the tape.
   1.4. Run \(M\) on \(w\) for \(|x|\) steps or until \(M\) naturally halts.
   1.5. If \(M\) naturally halted, then loop.
   1.6. Else halt.
2. Return \(<M\#>\).

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

- \(<M, w> \in \neg H\): No matter how long \(x\) is, \(M\) will not halt in \(|x|\) steps. So, for all inputs \(x\), \(M\#\) makes it to step 1.6. So it halts on everything. Oracle accepts.
- \(<M, w> \notin \neg H\): \(M\) halts on \(w\) in \(n\) steps. On inputs of length less than \(n\), \(M\#\) makes it to step 1.6 and halts. But on all inputs of length \(n\) or greater, \(M\#\) will loop in step 1.5. Oracle does not accept.
EqTMs = \{<M_a, M_b> : L(M_a) = L(M_b)\}

We’ve already shown it’s not in D.

Now we show it’s also not in SD.
EqTMs = \{<M_a, M_b> : L(M_a) = L(M_b)\}

\neg H = \{<M, w> : \text{TM } M \text{ does not halt on } w\}

R

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

R(<M, w>) =
1. Construct the description <M#>:

2. Construct the description <M?>:


If Oracle exists, C = Oracle(R(<M, w>)) semidecides \neg H:
• <M, w> ∈ \neg H:
• <M, w> ∉ \neg H:
EqTMs = \{<M_a, M_b> : L(M_a) = L(M_b)\}

\[ R(<M, w>) = \]
  1. Construct the description \(<M\#>:\)
     1.1 Erase the tape.
     1.2 Write \(w\) on the tape.
     1.3 Run \(M\) on \(w\).
     1.4 Accept.
  2. Construct the description \(<M?>:\)
     1.1 Loop.
  3. Return \(<M\#, M?>.\)

If Oracle exists, \(C = Oracle(R(<M, w>))\) semidecides \(\neg H: M?\) halts on nothing.
- \(<M, w> \in \neg H: M\) does not halt on \(w\), so \(M\#\) gets stuck in step 1.3 and halts on nothing. Oracle accepts.
- \(<M, w> \notin \neg H: M\) halts on \(w\), so \(M\#\) halts on everything. Oracle does not accept.
The Details Matter

$L_1 = \{<M>: M \text{ has an even number of states}\}$.

$L_2 = \{<M>: |<M>| \text{ is even}\}$.

$L_3 = \{<M>: |L(M)| \text{ is even}\}$.

$L_4 = \{<M>: M \text{ accepts all even length strings}\}$.
The Details Matter

\[ L_1 = \{ <M>: M \text{ has an even number of states} \}. \]

\[ L_2 = \{ <M>: |<M>| \text{ is even} \}. \]

\[ L_3 = \{ <M>: |L(M)| \text{ is even} \}. \]

\( \neg H \geq_M L_3: \ R(<M, w>) = \)

1. Construct the description \(<M\#>\), where \(M\#(x)\) operates as follows:
   1.1 Copy the input \(x\) to another track for later.
   1.2 Erase the tape.
   1.3 Write \(w\) on the tape.
   1.4 Run \(M\) on \(w\).
   1.5 If \(x = \varepsilon\) then accept. Else loop.

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

- \(<M, w> \in \neg H:\)
- \(<M, w> \notin \neg H:\)
The Details Matter

$L_1 = \{ <M>: M \text{ has an even number of states} \}$.

$L_2 = \{ <M>: |<M>| \text{ is even} \}$.

$L_3 = \{ <M>: |L(M)| \text{ is even} \}$.

$L_4 = \{ <M>: M \text{ accepts all even length strings} \}$. 

The Details Matter

$L_1 = \{<M>: M \text{ has an even number of states}\}$.

$L_2 = \{<M>: |<M>| \text{ is even}\}$.

$L_3 = \{<M>: |L(M)| \text{ is even}\}$.

$L_4 = \{<M>: M \text{ accepts all even length strings}\}$

$\neg H \geq M L_4: R(<M, w>) =$

1. Construct the description $<M\#>$, where $M\#(x)$ operates as follows:
   1.1 Copy the input $x$ to another track for later.
   1.2 Erase the tape.
   1.3 Write $w$ on the tape.
   1.4 Run $M$ on $w$ for $|x|$ steps or until $M$ naturally halts.
   1.5 If $M$ halted naturally, then loop. Else accept.

2. Return $<M\#>$.

- $<M, w> \in \neg H$:
- $<M, w> \notin \neg H$: 
Accepting, Rejecting, Halting, and Looping

Consider:

$L_1 = \{<M, w>: M \text{ rejects } w\}$.

$L_2 = \{<M, w>: M \text{ does not halt on } w\}$.

$L_3 = \{<M, w>: M \text{ is a deciding TM and rejects } w\}$.
\{<M, w>: M is a Deciding TM and Rejects w}\}

\[\neg H = \{<M, w>: \text{TM } M \text{ does not halt on } w\}\]

\[R\]

(\text{Oracle}) \quad \{<M, w>: M \text{ is a deciding TM and rejects } w\}

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

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 Reject.

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

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

- \(<M, w> \in \neg H:\)
- \(<M, w> \notin \neg H:\)

Problem:
\{<M, w>: M is a Deciding TM and Rejects w} \}

\[ H_\text{ALL} = \{<M>: \text{TM } M \text{ halts on } \Sigma^*\} \]

\[
R \quad (\text{Oracle}) \quad \{<M, w>: M \text{ is a deciding TM and rejects } w\}
\]

\[
R(<M>) = \\
\quad 1. \text{Construct the description } <M\#>, \text{ where } M\#(x) \text{ operates as follows:} \\
\quad \quad 1.1 \text{ Run } M \text{ on } x. \\
\quad \quad 1.2 \text{ Reject.} \\
\quad 2. \text{ Return } <M\#, \varepsilon>.
\]

If Oracle exists, \( C = \text{Oracle}(R(<M>)) \) semidecides \( H_\text{ALL} \):

- \( <M> \in H_\text{ALL} \): \( M\# \) halts and rejects all inputs. Oracle accepts.
- \( <M> \notin H_\text{ALL} \): There is at least one input on which \( M \) doesn’t halt. So \( M\# \) is not a deciding TM. Oracle does not accept.

No machine to semidecide \( H_\text{ALL} \) can exist, so neither does Oracle.
What About These?

\[L_1 = \{a\}.\]

\[L_2 = \{<M> : M \text{ accepts } a\}.\]

\[L_3 = \{<M> : L(M) = \{a\}\}.\]
What About These?

$L_1 = \{a\}$.

$L_2 = \{<M> : M \text{ accepts } a\}$.

$L_3 = \{<M> : L(M) = \{a\}\}$.

$\neg H \geq_M L_3$: $R(<M, w>) =$

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

- $<M, w> \in \neg H$: 
- $<M, w> \notin \neg H$: 

\{<M_a, M_b> : \varepsilon \in L(M_a) - L(M_b)\}
\{<M_a, M_b> : \varepsilon \in L(M_a) - L(M_b)\}

R( ) =

Return \(<M?, M#>\).

\(<M, w> \not\in \neg H: L(M?) - L(M#) =

\(<M, w> \not\in \neg H: L(M?) - L(M#) =
\[ \{<M_a, M_b> : \varepsilon \in L(M_a) - L(M_b)\} \]

\( R \) is a reduction from \( \neg H \). \( R(<M, w>) = \\
1. Construct the description of \( M\#(x) \) that operates as follows:
   1.1. Erase the tape.
   1.2. Write \( w \).
   1.3. Run \( M \) on \( w \).
   1.4. Accept.
2. Construct the description of \( M?(x) \) that operates as follows:
   2.1. Accept.
3. Return \( <M?, M\#> \).

If \( \text{Oracle} \) exists and semidecides \( L \), \( C = \text{Oracle}(R(<M, w>)) \)
semidecides \( \neg H \): \( M? \) accepts everything, including \( \varepsilon \). So:

- \( <M, w> \in \neg H : L(M?) - L(M\#) = 
- \( <M, w> \notin \neg H : L(M?) - L(M\#) = \)
<table>
<thead>
<tr>
<th><strong>The Problem View</strong></th>
<th><strong>The Language View</strong></th>
<th><strong>Status</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Does TM $M$ have an even number of states?</td>
<td>${&lt;M&gt; : M$ has an even number of states$}$</td>
<td>D</td>
</tr>
<tr>
<td>Does TM $M$ halt on $w$?</td>
<td>$H = {&lt;M, w&gt; : M$ halts on $w$}</td>
<td>SD/D</td>
</tr>
<tr>
<td>Does TM $M$ halt on the empty tape?</td>
<td>$H_\varepsilon = {&lt;M&gt; : M$ halts on $\varepsilon$}</td>
<td>SD/D</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>
<td>SD/D</td>
</tr>
<tr>
<td>Does TM $M$ halt on all strings?</td>
<td>$H_{\text{ALL}} = {&lt;M&gt; : M$ halts on $\Sigma^*$}</td>
<td>$\neg$SD</td>
</tr>
<tr>
<td>Does TM $M$ accept $w$?</td>
<td>$A = {&lt;M, w&gt; : M$ accepts $w$}</td>
<td>SD/D</td>
</tr>
<tr>
<td>Does TM $M$ accept $\varepsilon$?</td>
<td>$A_\varepsilon = {&lt;M&gt; : M$ accepts $\varepsilon$}</td>
<td>SD/D</td>
</tr>
<tr>
<td>Is there any string that TM $M$ accepts?</td>
<td>$A_{\text{ANY}} {&lt;M&gt; : \text{there exists at least one string that TM } M \text{ accepts}}$</td>
<td>SD/D</td>
</tr>
<tr>
<td>Does TM $M$ accept all strings?</td>
<td>$A_{\text{ALL}} = {&lt;M&gt;: L(M) = \Sigma^*} $</td>
<td>$\neg$ SD</td>
</tr>
<tr>
<td>---------------------------------</td>
<td>---------------------------------</td>
<td>----------</td>
</tr>
<tr>
<td>Do TMs $M_a$ and $M_b$ accept the same languages?</td>
<td>$\text{EqTM}s = {&lt;M_a, M_b&gt;: L(M_a) = L(M_b)} $</td>
<td>$\neg$ SD</td>
</tr>
<tr>
<td>Does TM $M$ not halt on any string?</td>
<td>$H_{\neg \text{ANY}} = {&lt;M&gt;: \text{there does not exist any string on which } M \text{ halts}} $</td>
<td>$\neg$ SD</td>
</tr>
<tr>
<td>Does TM $M$ not halt on its own description?</td>
<td>${&lt;M&gt;: \text{TM } M \text{ does not halt on input } &lt;M&gt;} $</td>
<td>$\neg$ SD</td>
</tr>
<tr>
<td>Is TM $M$ minimal?</td>
<td>$\text{TM}_{\text{MIN}} = {&lt;M&gt;: M \text{ is minimal}} $</td>
<td>$\neg$ SD</td>
</tr>
<tr>
<td>Is the language that TM $M$ accepts regular?</td>
<td>$\text{TM}_{\text{reg}} = {&lt;M&gt;: L(M) \text{ is regular}} $</td>
<td>$\neg$ SD</td>
</tr>
<tr>
<td>Does TM $M$ accept the language $A^nB^n$?</td>
<td>$A_{\text{anbn}} = {&lt;M&gt;: L(M) = A^nB^n} $</td>
<td>$\neg$ SD</td>
</tr>
</tbody>
</table>
Language Summary

IN
Semideciding TM
Enumerable
Unrestricted grammar

Deciding TM
Lexico. enum
L and ¬L in SD

CF grammar
PDA
Closure

Regular Expression
FSM

OUT
Reduction

Diagonalize
Reduction

Pumping
Closure

Pumping
Closure

SD
H

D
A^nB^nC^n

Context-Free
A^nB^n

Regular
a*b*

Language Summary

IN
Semideciding TM
Enumerable
Unrestricted grammar

Deciding TM
Lexico. enum
L and ¬L in SD

CF grammar
PDA
Closure

Regular Expression
FSM

OUT
Reduction

Diagonalize
Reduction

Pumping
Closure

Pumping
Closure