The Church-Turing Thesis

Chapter 18
Can We Do Better?

FSM $\Rightarrow$ PDA $\Rightarrow$ Turing machine

Is this the end of the line?

There are still problems we cannot solve:

- There is a countably infinite number of Turing machines since we can lexicographically enumerate all the strings that correspond to syntactically legal Turing machines.
- There is an uncountably infinite number of languages over any nonempty alphabet.
- So there are more languages than there are Turing machines. But can we do better by creating some new formalism?

Restate the question:
- Is there any computational algorithm that cannot be implemented by Turing machine? Then, if there is, can we find some more powerful model in which we could implement that algorithm?
What Can Algorithms Do?

• During 1\textsuperscript{st} third of 20\textsuperscript{th} century, a group of influential mathematicians focused on developing a \textbf{completely formal basis} for mathematics
  – Principia Mathematica (Whitehead and Russell 1920), most influential work on logic ever
  – Hibert’s program (David Hibert), to find a \textbf{complete and consistent set of axioms} for all of mathematics

• The continuation and ultimate success of this line of work depended on positive answers to two key questions:

1. Can we make all true statements theorems?

2. Can we decide whether a statement is a theorem?
Bertrand Russell (1872 – 1970)

English philosopher, logician, mathematician …
• In 1950, awarded Nobel in literature
• Russell’s family, one of the most influential in England, politically, land-owning, Earl, Duke, Baron, prime minister
• Has a (even more) famous student, Ludwig Wittgenstein
• The two are widely referred to as the greatest philosophers of last century

Russell’s paradox: showed that the naive set theory leads to a contradiction.

A set containing exactly the sets that are not members of themselves
  • whether the set contain itself?

• One applied version: Barber paradox

A barber shaves all and only those men in town who do not shave themselves.
  • construct a set containing men that the barber shaves. whether the barber himself should be in the set?

• Applications:
  • By Kurt Gödel, in incompleteness theorem by formalizing the paradox
  • By Turing, in undecidability of the Halting problem (and with that the Entscheidungsproblem) by using the same trick
David Hilbert (1862 – 1943)

German mathematician
• One of the most influential and universal mathematicians of the 19th and early 20th centuries.
• Hibert’s 23 unsolved problems: set the course for much of the mathematical research of the 20th century.
  • International Congress of Mathematicians, Paris, 1900
  • most successful and deeply considered compilation of open problems ever produced by an individual mathematician

1. Can we make all true statements theorems?
• 2\textsuperscript{nd} problem. Negative answer by Godel, incompleteness theorems, showing that Hibert’s program to find a complete and consistent set of axioms for all of mathematics is impossible
• On his tombstone, one can read his epitaph, the famous lines he had spoken at the end of his retirement address to the society of German Scientists and Physicians in 1930:
  We must know.
  We will know.

The day before, Godel, in a joint conference, tentatively announced the first expression of his incompleteness theorem, making Hilbert "somewhat angry".

2. Can we decide whether a statement is a theorem?
• The Entscheidungsproblem (German: decision problem), 1928.
• Negative answer from Alonzo Church and Alan Turing
Gödel’s Incompleteness Theorem

Kurt Gödel showed, in the proof of his Incompleteness Theorem [Gödel 1931], that the answer to question 1 is no. In particular, he showed that there exists no decidable axiomatization of Peano arithmetic that is both consistent and complete.

1. Can we make all true statement theorems?

• These theorems ended a half-century of attempts, beginning with the work of Frege and culminating in Principia Mathematica and Hilbert's formalism, to find a set of axioms sufficient for all mathematics.

• The incompleteness theorems also imply that not all mathematical questions are computable.

  • Leonid A. Levin: Cook-Levin theorem
Kurt Gödel

- 1906 – 1978. One of the greatest logicians of all time
- Gödel and Einstein … were known to take long walks together to and from the Institute for Advanced Study. … toward the end of his life Einstein confided that his "own work no longer meant much, that he came to the Institute merely…to have the privilege of walking home with Gödel."
- 1947, Einstein … accompanied Gödel to his U.S. citizenship exam, where they acted as witnesses. Gödel had confided in them that he had discovered an inconsistency in the U.S. Constitution, one that would allow the U.S. to become a dictatorship
- In later life, suffered periods of mental instability… fear of being poisoned; wouldn't eat unless his wife tasted his food for him. Late in 1977, wife was hospitalized for six months. In her absence, he refused to eat, eventually starving himself to death. He weighed 65 pounds when he died.
The Entscheidungsproblem

2. Can we decide whether a statement is a theorem?

Equivalent ways to state the problem:

Does there exist an algorithm to decide, given an arbitrary sentence \( w \) in first order logic, whether \( w \) is valid?

Given a set of axioms \( A \) and a sentence \( w \), does there exist an algorithm to decide whether \( w \) is entailed by \( A \)?

Given a set of axioms, \( A \), and a sentence, \( w \), does there exist an algorithm to decide whether \( w \) can be proved from \( A \)?
The Entscheidungsproblem

In 1936 and 1937, Church and Turing published independent papers showing that it is impossible to decide algorithmically whether statements in arithmetic are true or false, and thus a general solution to the Entscheidungsproblem is impossible.

This result is now known as Church's Theorem or the Church–Turing Theorem (not to be confused with the Church–Turing thesis).

To answer the question, in any of these forms, requires formalizing the definition of an algorithm:

- Even though algorithms have had a long history in mathematics, the notion of algorithm was not defined precisely before this.
- Church: lambda calculus
- Turing: Turing machines
- Turing proved that the two are equivalent.
  - Perhaps this observation can be extended (to church-turing thesis)
Alonzo Church

- American mathematician and logician
- Major contributions to mathematical logic and foundations of mathematical logic
- Church theorem (Church-Turing theorem)
- Church thesis (Church-Turing thesis)
- Lambda calculus
- Taught at Princeton, 1929–1967

Church's doctoral students were an extraordinarily accomplished lot, including
  - Stephen Kleene
  - Michael O. Rabin
  - Dana Scott
  - Alan Turing.

1903 – 1995
Alan Turing

1912 – 1954

One of the 100 Most Important People of the 20th Century

- For his role in the creation of the modern computer
- "The fact remains that everyone who taps at a keyboard, opening a spreadsheet or a word-processing program, is working on an incarnation of a Turing machine."

- Turing machine, influential formalization of the concept of the algorithm and computation
- Turing test, influential in AI

1936 – 1938, PhD, Princeton, under Alonzo Church

- Then, back to Cambridge, attended lectures by Ludwig Wittgenstein about the foundations of mathematics.
  - Ludwig Wittgenstein, student of Bertrand Russell at Cambridge, the two are widely referred to as the greatest philosophers of last century
Church's Thesis (Church-Turing Thesis)

All formalisms powerful enough to describe everything we think of as a computational algorithm are equivalent.

Implication: we should not expect to find some other reasonable computational model that is more powerful
- that can solve problems not solvable by Turing machines
- that would provide positive answers to the 2 questions

Profound philosophical implications

This isn’t a formal statement, so we can’t prove it. But today the thesis has near-universal acceptance.
- Many different computational models have been proposed and they all turn out to be equivalent.
The Church-Turing Thesis

Examples of equivalent formalisms:

- Modern computers (with unbounded memory)
- Lambda calculus
- Partial recursive functions
- Tag systems (FSM plus FIFO queue)
- Unrestricted grammars:
  \[ aS_\alpha \rightarrow B \]
- Post production systems
- Markov algorithms
- Conway’s Game of Life
- One dimensional cellular automata
- DNA-based computing
- Lindenmayer systems
The Unsolvability of the Halting Problem

Chapter 19
What We Can Compute

● Until a bit before the middle of the 20th century, western mathematicians believed that it would eventually be possible to prove any true mathematical statement, and to define an algorithm to solve any clearly stated mathematical problem

● Had they been right, our work would be done.

● But, they were wrong. There are well-defined problems for which no Turing machine exists.
  ● According to Church-Turing thesis, no other formalism is more powerful than Turing machines.

● Now, prove one of the most philosophically important theorems of the theory of computation: There is a specific problem (halting problem) that is algorithmically unsolvable.
  ● demonstrates that computers are limited in a fundamental way
  ● shows the limits of what we can compute
Languages and Machines

SD
Recursively enumerable

D
Recursive

Context-Free Languages

Regular Languages
\( \text{reg exps} \)
\( \text{FSMs} \)

\( \text{cfgs} \)
\( \text{PDAs} \)

unrestricted grammars

Turing Machines
A TM $M$ with input alphabet $\Sigma$ \textit{decides} a language $L \subseteq \Sigma^*$ iff, for any string $w \in \Sigma^*$,
\begin{itemize}
    \item if $w \in L$ then $M$ accepts $w$, and
    \item if $w \notin L$ then $M$ rejects $w$.
\end{itemize}

A language $L$ is \textit{decidable} (in D) iff there is a Turing machine $M$ that decides it.

A TM $M$ with input alphabet $\Sigma$ \textit{semidecides} $L$ iff for any string $w \in \Sigma^*$,
\begin{itemize}
    \item if $w \in L$ then $M$ accepts $w$
    \item if $w \notin L$ then $M$ does not accept $w$. $M$ may reject or loop.
\end{itemize}

A language $L$ is \textit{semidecidable} (in SD) iff there is a Turing machine that semidecides it.
The Language $H$

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

- $H$ is easy to state and understand.
- of great practical importance since a program to decide $H$ could be a very useful part of a program-correctness checker

**Theorem:** The language:

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

- is semidecidable, but
- is not decidable
3x + 1 Problem

Does times3 always halt?

times3(x: positive integer) =
While x ≠ 1 do:
   If x is even then x = x/2.
   Else x = 3x + 1

<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>25</td>
<td>29</td>
<td>34</td>
<td>40</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>76</td>
<td>88</td>
<td>17</td>
<td>20</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>38</td>
<td>44</td>
<td>52</td>
<td>10</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>19</td>
<td>22</td>
<td>26</td>
<td>5</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>58</td>
<td>11</td>
<td>13</td>
<td>16</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

• It is conjectured that, for any positive integer, times3 halts. But so far, no proof, no counterexample.
• So there appear to be programs whose halting behavior is difficult to determine.
H is Semidecidable

**Lemma:** The language:

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

is semidecidable.

**Proof:** The TM \( M_{SH} \) semidecides \( H \):

\[ M_{SH}(<M, w>) = \]

1. Run \( M \) on \( w \).

\( M_{SH} \) halts iff \( M \) halts on \( w \). Thus \( M_{SH} \) semidecides \( H \).

\( M_{SH} \) is a universal Turing machine
The Unsolvability of the Halting Problem

Lemma: The language:

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

is not decidable.

Proof: We assume \( H \) is decidable and obtain a contradiction. Suppose \( M_H \) is a decider for \( H \). Then \( M_H \) would implement the specification:

\[ M_H(<M, w>) = \begin{cases} \text{accept} & \text{if } M \text{ halts on input } w \\ \text{reject} & \text{else} \end{cases} \]
Now we construct a new TM $M_D$ such that

$$M_D(<M>) = \begin{cases} \text{loop forever} & \text{if } M \text{ halts} \\ \text{halt} & \text{else} \end{cases}$$

- Note, we are able to construct $M_D$ that does the opposite of $M$ depends on the existence of $M_H$, that is, H is decidable.
- We can use $M_H$ as a subroutine to construct $M_D$.

Now, what happens if we run $M_D$ with $<M_D>$?

$$M_D(<M_D>) = \begin{cases} \text{loop forever} & \text{if } M_D \text{ halts} \\ \text{halt} & \text{else} \end{cases}$$

Contradiction established.
Recall Russell’s Paradox

A set containing exactly the sets that are not members of themselves
• whether the set contain itself?

• One applied version: Barber paradox

A barber shaves all and only those men in town who do not shave themselves.
• construct a set containing men that the barber shaves. whether the barber himself should be in the set?

• Russell’s paradox shows the undecidability of the membership of an arbitrary set.

• The same trick is used to show H is not decidable.
Viewing the Halting Problem as Diagonalization

- Lexicographically enumerate Turing machines.
- Let 1 mean halting, blank mean non halting.

<table>
<thead>
<tr>
<th></th>
<th>$i_1$</th>
<th>$i_2$</th>
<th>$i_3$</th>
<th>...</th>
<th>$&lt;M_D&gt;$</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>machine$_1$</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>machine$_2$</td>
<td></td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>machine$_3$</td>
<td></td>
<td></td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$M_D$</td>
<td>1</td>
<td></td>
<td></td>
<td>?</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- If we claim that $M_H$ exists, we are claiming that it can compute the correct value for any cell in this table on demand.

- $M_D$ computes the opposite of the diagonal entries:

```
|       | 1     |       | ?     | 1   |
```

What value should occur in $?$? Contradiction arises as the entry must be the opposite of itself!
Undecidability of the Halting Problem

To show the Halting Problem can't be algorithmically solved,
We use diagonalization, a proof method that's evolved
Where Turing Machines are asked to run in modes of simulation,
On codes that represent themselves, an auto-copulation.

Suppose a TM we'll call $Halt$, with input that's a code
Of another machine $M$, and in addition is showed
An input $x$ to $M$, whence $Halt$ says (and it's always right)
Whether $M$ did halt on $x$. (But how? Divine insight?)

Let TM $D$, with input that's the code of machine $A$
Use $Halt$ to find if $A$ will halt on its own code, then say,
``If the answer's `no' I'll output 1, else fight the urge
To give another answer''-- better then that $D$ diverge.

Rather than recount the proof's denouement, we'll be prudent,
And leave the rest to you, dear reader, the ever-able student.
When $Halt$ is given the code of $D$, it simply cannot know
If $D$ does halt on its own code--but that's for you to show.

The trick is to diagonalize--$D$ changed the bit returned
By $Halt$, thus contradicts its work, as its output is spurned.
No matter how a TM solves the Halting Problem, see,
Diagonalization makes a counter-example. Q.E.D.
Implications of the Undecidability of H

- H is far more than an anomaly. It is the key to the fundamental distinction between the classes D and SD.

**Theorem:** If H were in D then every SD language would be in D.

**Proof:** Let $L$ be any SD language. There exists a TM $M_L$ that semidecides it.

If H were also in D, then there would exist an $O$ that decides it.

To decide whether $w$ is in $L(M_L)$:

$M'(w: \text{string}) =$

1. Run $O$ on $\langle M_L, w \rangle$.
2. If $O$ accepts (i.e., $M_L$ will halt), then:
   1. Run $M_L$ on $w$.
   2. If it accepts, accept. Else reject.
3. Else reject.

So, if H were in D, all SD languages would be.

T or F: If $\neg H$ were in D then every SD language would be in D.
Back to the Entscheidungsproblem

- Having defined the Turing machine, Turing went on to show the unsolvability of the halting problem. He then used that result to show the unsolvability of the Entscheidungsproblem.

**Theorem:** The Entscheidungsproblem is unsolvable.

**Proof:** (Due to Turing)
1. If we could solve the problem of determining whether a given Turing machine ever prints the symbol 0, then we could solve the problem of determining whether a given Turing machine halts.
2. But we can’t solve the problem of determining whether a given Turing machine halts, so neither can we solve the problem of determining whether it ever prints 0.
3. Given a Turing machine $M$, we can construct a logical formula $F$ that is true iff $M$ ever prints the symbol 0.
4. If there were a solution to the Entscheidungsproblem, then we would be able to determine the truth of any logical sentence, including $F$ and thus be able to decide whether $M$ ever prints the symbol 0.
5. But we know that there is no procedure for determining whether $M$ ever prints 0.
6. So there is no solution to the Entscheidungsproblem.
Decidable and Semidecidable Languages

Chapter 20
D and SD Languages

- SD Languages
- Regular Languages
- Context-Free Languages
- D Languages
Every CF Language is in D

**Theorem:** The set of context-free languages is a *proper* subset of D.

**Proof:**

- Every context-free language is decidable, so the context-free languages are a subset of D.
- There is at least one language, \( A^nB^nC^n \), that is decidable but not context-free.

So the context-free languages are a *proper* subset of D.
Decidable and Semidecidable Languages

Almost every obvious language that is in SD is also in D:

- \( A^n B^n C^n = \{a^m b^n c^n, \ n \geq 0\} \)
- \( \{wcw, \ w \in \{a, b\}^*\} \)
- \( \{ww, \ w \in \{a, b\}^*\} \)
- \( \{w = x^* y^* z: \ x, y, z \in \{0, 1\}^* \) and, when \( x, y, \) and \( z \) are viewed as binary numbers, \( xy = z\}\)

But there are languages that are in SD but not in D:

- \( H = \{<M, w>: M \text{ halts on input } w\} \)
D and SD

1. D is a subset of SD. In other words, every decidable language is also semidecidable.
2. There exists at least one language that is in SD/D, the donut in the picture.
3. There exist languages that are not in SD. In other words, the gray area of the figure is not empty.
Languages That Are Not in SD

**Theorem:** There are languages that are not in SD.

**Proof:** Assume any nonempty alphabet $\Sigma$.

**Lemma:** There is a countably infinite number of SD languages over $\Sigma$.

**Proof:**

**Lemma:** There is an uncountably infinite number of languages over $\Sigma$.

So there are more languages than there are languages in SD. Thus there must exist at least one language that is in $\neg$SD.
Closure Under Complement

Regular languages are closed under complement. Context free languages are not. How about D and SD?

**Theorem:** The set D is closed under complement. 

**Proof:** (by construction) …

**Theorem:** The set SD is not closed under complement.
Closure Properties of D and SD

D is closed under:
  • union
  • Intersection
  • complement
  • set difference
  • concatenation,
  • Kleene star

SD is closed under:
  • union
  • Intersection
  • concatenation,
  • Kleene star
D and SD Languages

**Theorem:** A language is in D iff both it and its complement are in SD.

**Proof:**

- **L** in D implies **L** and **¬L** are in SD:
  - **L** is in SD because D ⊂ SD.
  - D is closed under complement
  - So **¬L** is also in D and thus in SD.

- **L** and **¬L** are in SD implies **L** is in D:
  - **M**₁ semidecides **L**.
  - **M**₂ semidecides **¬L**.
  - To decide **L**:
    - Run **M**₁ and **M**₂ in parallel on **w**.
    - Exactly one of them will eventually accept.


A Language that is Not in SD

**Theorem:** The language $\neg H = \{<M, w> : \text{TM } M \text{ does not halt on input string } w\}$ is not in SD.

**Proof:**
- $H$ is in SD.
- If $\neg H$ were also in SD then $H$ would be in D.
- But $H$ is not in D.
- So $\neg H$ is not in SD.
Enumerate a Language

So far, we have defined a language by specifying either a grammar that can generate it or a machine that can accept it. But it’s also possible to specify a machine that is a generator.

Enumerate means list.

We say that Turing machine $M$ enumerates the language $L$ iff, for some fixed state $p$ of $M$:

$$L = \{w : (s, \varepsilon) \vdash^*_{M^*} (p, w)\}.$$

A language is Turing-enumerable iff there is a Turing machine that enumerates it.
Theorem: A language is SD iff it is Turing-enumerable.

Turing Enumerable = SD
Lexicographic Enumeration

M lexicographically enumerates L iff M enumerates the elements of L in lexicographic order.

A language L is lexicographically Turing-enumerable iff there is a Turing machine that lexicographically enumerates it.

Example: $A^nB^nC^n = \{a^n b^n c^n : n \geq 0\}$

Lexicographic enumeration:
Lexicographically Enumerable = D

Theorem: A language is in D iff it is lexicographically Turing-enumerable.
Decidability and Undecidability Proofs

Sections 21.1 – 21.3
## Some 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>$H_\varepsilon = { &lt;M&gt; : M \text{ halts on } \varepsilon }$</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>$\text{EqTMs} = { &lt;M_a, M_b&gt; : L(M_a) = L(M_b) }$</td>
</tr>
<tr>
<td>Is the language that TM $M$ accepts regular?</td>
<td>$\text{TM}_{\text{reg}} = { &lt;M&gt; : L(M) \text{ is regular} }$</td>
</tr>
</tbody>
</table>
Reduction is Ubiquitous

- Calling Jen

```
Call Jen

Get hold of Jim
```

- Reduce a problem $P_{old}$ to another problem $P_{new}$. The solution for $P_{new}$ can be used to build a solution for $P_{old}$
- Usage 1: Known solution for $P_{new}$, obtain a solution for $P_{old}$
- Usage 2: Known insolvability of $P_{old}$, prove insolvability of $P_{new}$
- Prove by contradiction: if $P_{new}$ were solvable, we can use its solution to build a solution for $P_{old}$. But $P_{old}$ is not solvable, so $P_{new}$ can’t be solvable
Reduction

A reduction $R$ from $L_{\text{old}}$ to $L_{\text{new}}$ consists of one or more Turing machines such that:

If there exists a Turing machine Oracle that decides (or semidecides) $L_{\text{new}}$, then the Turing machines in $R$ can be composed with Oracle to build a deciding (or a semideciding) Turing machine for $L_{\text{old}}$.

$$P_{\text{old}} \leq P_{\text{new}}$$ means that $P_{\text{old}}$ is reducible to $P_{\text{new}}$. 
Using Reduction

\( P_{\text{old}} \leq P_{\text{new}} \) means that \( P_{\text{old}} \) is reducible to \( P_{\text{new}} \)

By definition:
1. If \( P_{\text{new}} \) is in D, then \( P_{\text{old}} \) must be in D
2. If \( P_{\text{new}} \) is in SD, then \( P_{\text{old}} \) must be in SD

Actual usage:
1. Known \( P_{\text{old}} \) is not in D, we can show \( P_{\text{new}} \) is not in D
2. Known \( P_{\text{old}} \) is not in SD, we can show \( P_{\text{new}} \) is not in SD
   • For both, no need to care about the efficiency of \( R \)

Can also be used in complexity to show NP-hardness:
3. Known \( P_{\text{old}} \) is NP-hard, we can show \( P_{\text{new}} \) is NP-hard
   • Need to care about the efficiency of \( R \)

Common mistake: doing reduction backwards.
Common mistake

Useless backwards reduction $P_{\text{new}} \leq P_{\text{old}}$

1. Known $P_{\text{old}}$ is not in D, we CANNOT show $P_{\text{new}}$ is not in D
   - example: $\{a\} \leq H$, $H$ is not in D, but $\{a\}$ is in D.

   Why $\{a\} \leq H$?

   **Theorem**: If $H$ were in D then every SD language would be in D (slide 26) i.e., every SD language is reducible to $H$.

2. Known $P_{\text{old}}$ is not in SD, we CANNOT show $P_{\text{new}}$ is not in SD
   - Example: $\{a\} \leq \neg H$. $\neg H$ is not in SD but $\{a\}$ is in SD.

   Why $\{a\} \leq \neg H$?
   Because if $\neg H$ is in D, then $H$ must be in D and every SD language must be in D based on the above theorem.
To Use Reduction for Undecidability

1. Choose a language $L_{old}$:
   - that is already known not to be in $D$, and
   - that can be reduced to $L_{new}$ (i.e., there can be a deciding machine for $L_{old}$ if there existed a deciding machine for $L_{new}$), and
   - the reduction is as straightforward as possible.

2. Define the reduction $R$.

3. Describe the composition $C$ of $R$ with Oracle (the machine that hypothesize decides $L_{new}$).

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

The most straightforward way of reduction is to transform instances of $L_{old}$ into instances of $L_{new}$.

$L_{old}$ is *mapping reducible* to $L_{new}$ ($L_{old} \leq_M L_{new}$) iff there exists some computable function $f$ such that:

$$\forall x \in \Sigma^* \ (x \in L_{old} \iff f(x) \in L_{new})$$

To decide whether $x$ is in $L_{old}$, we transform it, using $f$, into a new object and ask whether that object is in $L_{new}$.

If $L_{old} \leq_M L_{new}$, $C(x) = Oracle(R(x))$ will decide $L_{old}$. 

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.
• 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.
$H_{ANY} = \{ \langle M \rangle : \text{there exists at least one string on which TM } M \text{ halts} \}$

**Theorem:** $H_{ANY}$ 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?
\[ T(<M>) = \]

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

\[
\begin{align*}
\varepsilon & \ [1] \\
\varepsilon & \ [2] \ a \ [1] \\
\varepsilon & \ [3] \ a \ [2] \ b \ [1] \\
\varepsilon & \ [4] \ a \ [3] \ b \ [2] \ aa \ [1] \\
\varepsilon & \ [5] \ a \ [4] \ b \ [3] \ aa \ [2] \ ab \ [1] \\
\varepsilon & \ [6] \ a \ [5] \ aa \ [3] \ ab \ [2] \ ba \ [1]
\end{align*}
\]

2. If any instance of \( M \) halts, halt and accept.

\( T \) will accept iff \( M \) halts on at least one string. So \( T \) semidecides \( H_{\text{ANY}} \).
$H_{ANY}$ is not in D

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

$R$

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

$R(<M, w>) =$

1. Construct $<M#>$, where $M#(x)$ 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 $H$:

$\bullet$ $R$ can be implemented as a Turing machine.

$\bullet$ $C$ is correct: The only string on which $M#$ can halt is $w$. So:

$\bullet$ $<M, w> \in H$: $M$ halts on $w$. So $M#$ halts on $w$. There exists at least one string on which $M#$ halts. $Oracle(<M#>)$ accepts.

$\bullet$ $<M, w> \notin H$: $M$ does not halt on $w$, so neither does $M#$. So there exists no string on which $M#$ halts. $Oracle(<M#>)$ rejects.

But no machine to decide $H$ can exist, so neither does Oracle.
H_{\text{ANY}} is not in D

To obtain a contradiction, suppose Oracle is a decider for H_{\text{ANY}}. We will use this to produce a decider C for H (which we know does not exist).

Given input M and w, define C(M, w) with the following behavior: C creates a Turing machine $M#$ that accepts only if the input string to $M#$ is w and M halts on input w, and does not halt otherwise. The decider C can now evaluate Oracle($M#$) to check whether the language accepted by $M#$ is non-empty ($M#$ halts on some string).

If Oracle accepts $M#$, then the language accepted by $M#$ is nonempty, so M does halt on input w, so C can accept. If Oracle rejects $M#$, then the language accepted by $M#$ is empty, so in particular M does not halt on input w, so C can reject. Thus, if we had a decider Oracle for H_{\text{ANY}}, we would be able to produce a decider C for the halting problem H(M, w). Since we know that such a C cannot exist, it follows that the language H_{\text{ANY}} is also undecidable.
Proof: We show that $H_{\text{ANY}}$ is not in $D$ by reduction from $H$:

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

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

$R(<M, w>) =$
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. $\text{Oracle}(<M#>)$ 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. $\text{Oracle}(<M#>)$ rejects.

But no machine to decide $H$ can exist, so neither does Oracle.
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 (languages of Turing machines) 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 SD, i.e., $P$ is a property of languages of Turing machines.
- 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\}.
Applying Rice’s Theorem

Rice's theorem applies to languages, not machines. So, for example, the following properties of machines are decidable:

- \( M \) contains an even number of states
- \( M \) has an odd number of symbols in its tape alphabet

Of course, we need a way to define a language. We'll use machines to do that, but the properties we'll deal with are properties of \( L(M) \), not of \( M \) itself.
Given a TM $M$, is $L(M)$ Regular?

The problem: Is $L(M)$ regular? (or, giving $M$, does $M$ only accept some regular language?)

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:
  - $P(a^*) = True$.
  - $P(A^nB^n) = False$. 
Non-SD Languages

There is an uncountable number of non-SD languages, but only a countably infinite number of TM’s (hence SD languages).
∴ 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
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)
Theorem: $H_{\neg\text{ANY}} = \{<M> : \text{there does not exist any string on which TM } M \text{ halts}\}$ is not in SD.

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

We already know:
- $\neg H_{\neg\text{ANY}}$ is in SD.
- $\neg H_{\neg\text{ANY}}$ is not in D.

So $H_{\neg\text{ANY}}$ is not in SD because, if it were, then $H_{\text{ANY}}$ would be in D but it isn’t.
**Using Reduction**

**Theorem:** If there is a reduction $R$ from $L_{old}$ to $L_{new}$ and $L_{old}$ is not SD, then $L_{new}$ is not SD.

So, we must:
- Choose a language $L_{old}$ that is known not to be in SD.
- Hypothesize the existence of a semideciding TM Oracle.
\[ H_{\text{ALL}} = \{ <M> : \text{TM halts on } \Sigma^* \} \]

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

\[ R \]

(Oracle) \( \neg H_{\text{ALL}} = \{ <M> : \text{TM does not halt on } \Sigma^* \} \)

**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

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

\[ R \]

(?Oracle) \quad H_{\text{ALL}} = \{ \langle M \rangle : \text{TM halts on } \Sigma^* \}

**Reduction Attempt 1:** \( R(\langle M, w \rangle) = \)

1. Construct the description \( \langle M# \rangle \), 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 \( \langle M# \rangle \).

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

\( \bullet \) \( \langle M, w \rangle \in \neg H: M \) does not halt on \( w \), so \( M# \) gets stuck in step 1.3 and halts on nothing. Oracle does not accept.

\( \bullet \) \( \langle M, w \rangle \notin \neg H: M \) halts on \( w \), so \( M# \) halts on everything. Oracle accepts.
\[ H_{\text{ALL}} = \{ \langle M \rangle : \text{TM halts on } \Sigma^* \} \]

\( R(<M, w>) \) reduces \( \neg H \) to \( H_{\text{ALL}} \):
1. Construct the description \( \langle M\# \rangle \), 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 \( \langle M\# \rangle \).

If Oracle exists, \( C = \text{Oracle}(R(<M, w>)) \) semidecides \( \neg H \):
- \( \langle M, w \rangle \in \neg H \): 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.
- \( \langle M, w \rangle \notin \neg H \): \( M \) 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.
<table>
<thead>
<tr>
<th>The Problem View</th>
<th>The Language View</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>Does TM $M$ have an even number of states?</td>
<td>${&lt;M&gt; : M \text{ has an even number of states}}$</td>
<td>D</td>
</tr>
<tr>
<td>Does TM $M$ halt on $w$?</td>
<td>$H = {&lt;M, w&gt; : M \text{ halts on } w}$</td>
<td>SD/D</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>
<td>SD/D</td>
</tr>
<tr>
<td>Is there any string on which TM $M$ halts?</td>
<td>$H_{\text{ANY}} = {&lt;M&gt; : \text{there exists at least one string on which } M \text{ halts}}$</td>
<td>SD/D</td>
</tr>
<tr>
<td>Does TM $M$ halt on all strings?</td>
<td>$H_{\text{ALL}} = {&lt;M&gt; : M \text{ halts on } \Sigma^*}$</td>
<td>$\neg$SD</td>
</tr>
<tr>
<td>Does TM $M$ accept $w$?</td>
<td>$A = {&lt;M, w&gt; : M \text{ accepts } w}$</td>
<td>SD/D</td>
</tr>
<tr>
<td>Does TM $M$ accept $\epsilon$?</td>
<td>$A_\epsilon = {&lt;M&gt; : M \text{ accepts } \epsilon}$</td>
<td>SD/D</td>
</tr>
<tr>
<td>Is there any string that TM $M$ accepts?</td>
<td>$A_{\text{ANY}} {&lt;M&gt; : \text{there exists at least one string that } M \text{ accepts}}$</td>
<td>SD/D</td>
</tr>
<tr>
<td>Question</td>
<td>Definition</td>
<td>Solution</td>
</tr>
<tr>
<td>-------------------------------------------------------------------------</td>
<td>---------------------------------------------------------------------------</td>
<td>----------</td>
</tr>
<tr>
<td>Does TM $M$ accept all strings?</td>
<td>$A_{ALL} = {&lt;M&gt; : L(M) = \Sigma^*}$</td>
<td>$\neg SD$</td>
</tr>
<tr>
<td>Do TMs $M_a$ and $M_b$ accept the same languages?</td>
<td>$\text{EqTMs} = {&lt;M_a, M_b&gt; : L(M_a) = L(M_b)}$</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>$A_{anbn} = {&lt;M&gt; : L(M) = A^nB^n}$</td>
<td>$\neg SD$</td>
</tr>
</tbody>
</table>
Language Summary

**IN**
- Semideciding TM
- Enumerable
- Unrestricted grammar

**SD**
- H
- \(L\) and \(\neg L\) in SD

**OUT**
- Reduction

**D**
- \(A^nB^nC^n\)

**Context-Free**
- \(A^nB^n\)

**Regular**
- \(a^*b^*\)

**Closure**
- Regular Expression
- FSM

**Pumping**
- CF grammar
- PDA

**Diagonalize Reduction**
- Deciding TM
- Lexic. enum

**Lexic. enum Reduction**
- In SD