Lecture 12:
Oracles and Self-Reference
Midterms back at end of lecture!

Thanks for your feedback

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>mean</td>
<td>72.81632653</td>
</tr>
<tr>
<td>stddev</td>
<td>14.7939648</td>
</tr>
<tr>
<td>median</td>
<td>75</td>
</tr>
</tbody>
</table>
### Some interesting feedback

<table>
<thead>
<tr>
<th>Feedback</th>
<th>Number of students</th>
</tr>
</thead>
<tbody>
<tr>
<td>HW is too hard</td>
<td>1</td>
</tr>
<tr>
<td>Fewer proofs in lecture (more concepts)</td>
<td>4</td>
</tr>
<tr>
<td>Too fast recitation/lecture</td>
<td>7</td>
</tr>
<tr>
<td>Likes Ryan/slides/lectures</td>
<td>33</td>
</tr>
<tr>
<td>Dislikes Ryan/slides/lectures</td>
<td>0</td>
</tr>
<tr>
<td>Likes TAs and recitations</td>
<td>16</td>
</tr>
<tr>
<td>More lecture notes (besides lecs)</td>
<td>3</td>
</tr>
<tr>
<td>More Office Hours</td>
<td>3</td>
</tr>
<tr>
<td>More examples/practice in lec/rec</td>
<td>7</td>
</tr>
<tr>
<td>Likes HW</td>
<td>7</td>
</tr>
<tr>
<td>I dislike that I don’t have any criticism</td>
<td>3</td>
</tr>
<tr>
<td>Faster pset solutions/grades</td>
<td>4</td>
</tr>
<tr>
<td>Likes piazza+office hours</td>
<td>12</td>
</tr>
<tr>
<td>“Yeet this class”</td>
<td>1</td>
</tr>
</tbody>
</table>
Recognizability via Logic

Def. A decidable predicate $R(x,y)$ is a logical proposition about input strings $x, y \in \Sigma^*$ that is implementable by some TM $M$. That is,

$$
\text{for all } x, y, \quad R(x,y) \text{ is True } \Rightarrow \quad M(x,y) \text{ accepts} \\
R(x,y) \text{ is False } \Rightarrow \quad M(x,y) \text{ rejects}
$$

Think of $R$ as a function: $R: \Sigma^* \times \Sigma^* \rightarrow \{\text{True, False}\}$

EXAMPLES: $R(x,y) = \text{“xy has at most 100 zeroes”}$
$R(N,y) = \text{“TM N halts on y in at most 99 steps”}$
are both decidable predicates
Theorem: A language $A \subseteq \Sigma^*$ is recognizable if and only if there is a decidable predicate $R(x, y)$ such that:

$$A = \{ x \mid \exists y \in \Sigma^* \ R(x, y) \}$$

Proof: (1) If $A = \{ x \mid \exists y \ R(x, y) \}$ then $A$ is recognizable

Let $M$ be a TM implementing $R$.
Define a TM $M'(x)$: For all strings $y \in \Sigma^*$,
If $M(x, y)$ accepts, accept.
Then, $M'$ accepts exactly those $x$ s.t. $\exists y \ R(x, y)$ is true

(2) If $A$ is recognizable, then $A = \{ x \mid \exists y \ R(x, y) \}$

Suppose TM $M$ recognizes $A$.
Define $R(x, y)$ to be TRUE iff $M$ accepts $x$ in $|y|$ steps
Then, $M$ accepts $x \iff \exists y \ R(x, y)$
Example: \( L = \{ M \mid M \text{ accepts at least one string} \} \) is recognizable.

Want: decidable predicate \( R \) such that
\( L = \{ M \mid \exists y \in \Sigma^* \ R(M, y) \text{ is true} \} \)

Define \( R(M,(x,y)) = \text{“TM M accepts string x in } |y| \text{ steps”} \)
Note that \( R(M,(x,y)) \) is decidable!

Then: \( L = \{ M \mid \exists (x,y) \in \Sigma^* \ R(M, (x,y)) \text{ is true} \} \)

So \( L \) is recognizable!
Computability
With Oracles

*We do not condone smoking. Don’t do it. It’s bad. Kthxbye
Oracle Turing Machines

Is \((M, w)\) in \(A_{TM}\)? Yes!

\(q_{YES}\)

INPUT

INFINITE TAPE

Now leaving reality for a moment....
An oracle Turing machine $M$ is equipped with a set $B \subseteq \Gamma^*$ and a special oracle tape, on which $M$ may ask membership queries about $B$. Formally, $M$ enters a special state $q_?$ to ask a query and the TM receives a query answer in one step. [Formally, the transition function on $q_?$ is defined in terms of the entire oracle tape:

State $q_?$ changes to $q_{\text{YES}}$
if the string $y$ written on the oracle tape is in $B$,
else $q_?$ changes to $q_{\text{NO}}$]

This notion makes sense even if $B$ is not decidable!
How to Think about Oracles?

Think in terms of Turing Machine pseudocode!

An oracle Turing machine $M$ with oracle $B \subseteq \Gamma^*$ lets you include the following kind of if-then statement:

```
"if (z in B) then <do something>
else <do something else>"
```

where $z$ is some string defined earlier in pseudocode. We define the oracle TM to that it can always check the condition $(z \text{ in } B)$ in one step.

This notion makes sense even if $B$ is not decidable!
Deciding one problem with another

Definition: A is decidable with B if there is an oracle TM $M$ with oracle $B$ that accepts strings in $A$ and rejects strings not in $A$

Language $A$ "Turing-Reduces" to $B$

$A \leq_T B$
\( A_{TM} \) is decidable with \( HALT_{TM} \) \( (A_{TM} \leq_T HALT_{TM}) \)

We can decide if \( M \) accepts \( w \) using an ORACLE for the Halting Problem:

On input \( (M,w) \),

If \( (M,w) \) is in \( HALT_{TM} \) then run \( M(w) \) and output its answer.
else REJECT.
HALT<sub>TM</sub> is decidable with A<sub>TM</sub> (HALT<sub>TM</sub> \leq_T A<sub>TM</sub>)

On input (M,w), decide if M halts on w as follows:

1. If (M,w) is in A<sub>TM</sub> then ACCEPT

2. Else, swap the accept and reject states of M to get a machine M′. If (M′,w) is in A<sub>TM</sub> then ACCEPT

3. REJECT
$\leq_T$ \ versus \ $\leq_m$

**Theorem:** If $A \leq_m B$ then $A \leq_T B$

**Proof (Sketch):**

$A \leq_m B$ means there is a computable function $f : \Sigma^* \rightarrow \Sigma^*$, where for every $w$,

$$w \in A \iff f(w) \in B$$

To decide $A$ on an input $w$ with oracle $B$, just compute $f(w)$, then call $B$ on $f(w)$ and return answer.

**Theorem:** $\overline{\text{HALT}}_{TM} \leq_T \text{HALT}_{TM}$

$D(M,w)$: If $((M,w) \in \text{HALT}_{TM})$ then reject else accept

**Theorem:** $\overline{\text{HALT}}_{TM} \not\leq_m \text{HALT}_{TM}$ \hspace{1cm} Why?
Limitations on Oracle TMs!

The following problem cannot be decided by any TM with an oracle for the Halting Problem:

\[ \text{SUPERHALT} = \{ (M,x) \mid \text{TM } M, \text{ with an oracle for the Halting Problem, halts on } x \} \]

We can use the original proof by diagonalization!
Assume \(H\) (with \(\text{HALT}\) oracle) decides \(\text{SUPERHALT}\)

Define \(D(X) := \text{“if } H(X,X) \text{ (with } \text{HALT} \text{ oracle) accepts then LOOP, else ACCEPT.”} \)

\(D\) uses a \(\text{HALT}\) oracle to simulate \(H\)
But \(D(D)\) halts \(\Leftrightarrow H(D,D)\) accepts \(\Leftrightarrow D(D)\) loops...
(by assumption on \(H\)) \hspace{1cm} (by def of \(D\))
Limitations on Oracle TMs!

There is an infinite hierarchy of unsolvable problems!

*Given ANY oracle A, there is always a harder problem that cannot be decided with that oracle A*

SUPERHALT^0 = HALT = \{ (M,x) \mid M \text{ halts on } x \}.

SUPERHALT^1 = \{ (M,x) \mid M, \text{ with an oracle for } \text{HALT}_{\text{TM}}, \text{ halts on } x \}\}

SUPERHALT^n = \{ (M,x) \mid M, \text{ with an oracle for } \text{SUPERHALT}^{n-1}, \text{ halts on } x \}
A Puzzle About Oracles

Given three instances 
$$(M_1, w_1), (M_2, w_2), (M_3, w_3)$$

of the Halting Problem,

It’s easy to decide all three of them, using three oracle calls to HALT.

Can you decide $$(M_i, w_i) \in \text{HALT}$$ for all i, with only TWO oracle calls to HALT?
Self-Reference and the Recursion Theorem
Lemma: There is a computable function 
\( q : \Sigma^* \rightarrow \Sigma^* \) such that for every string \( w \), 
\( q(w) \) is the description of a TM \( P_w \) that on every input, prints out \( w \) and then accepts

“Proof” Define a TM Q:

![Diagram]

- Actual TM
- String encoding a TM
Theorem: There is a Self-Printing TM

Proof: First define a TM B which does this:

Now consider the TM that looks like this:

No explicit self-reference here!

QED
Another Way of Looking At It

Suppose in general we want to design a program that prints its own description. How?

“Print this sentence.”

Print two copies of the following, the second copy in quotes:

“Print two copies of the following, the second copy in quotes:”

\[ = B \]

\[ = P_B \]
The Recursion Theorem

Theorem: For every TM $T$ computing a function $t : \Sigma^* \times \Sigma^* \rightarrow \Sigma^*$ there is a Turing machine $R$ computing a function $r : \Sigma^* \rightarrow \Sigma^*$, such that for every string $w$,

$$r(w) = t(R, w)$$

\begin{align*}
(a, b) &\quad \rightarrow \quad T \quad \rightarrow \quad t(a, b) \\
\quad &\quad \rightarrow \quad R \quad \rightarrow \quad t(R, w)
\end{align*}
Proof: \[(a,b) \rightarrow T \rightarrow t(a,b)\]

**Define M:**

- Define $M$:
  - $B \rightarrow C \rightarrow T \rightarrow P_N \rightarrow N$
  - $w \rightarrow B \rightarrow w \rightarrow T \rightarrow w$

**Define R:**

- Define $R$:
  - $P_M \rightarrow M \rightarrow B \rightarrow T \rightarrow t(S,w)$
  - $w \rightarrow P_M \rightarrow M \rightarrow S \rightarrow w$

What is $S$?
Proof: \((a,b) \rightarrow T \rightarrow t(a,b)\)

Define M:

Define R:
Proof: \[(a,b) \rightarrow T \rightarrow t(a,b)\]

Define R:

\[t(S,w) = t(S,w)\]

\[S = C = R. \quad \text{QED}\]
FOO\textsubscript{x}(y) := Output x and halt.
BAR(M) := Output “N(w) = Run FOO\textsubscript{M} outputting M.
          Run M on (M, w)”
Q(N, w) := Run BAR(N) outputting S.
          Run T on (S, w)
R(w) := Run FOO\textsubscript{Q} outputting Q.
          Run BAR(Q) outputting S.
          Run T on (S, x)
Claim: S is a description of R itself!
S(w) = Run FOO\textsubscript{Q} outputting Q.
       Run Q on (Q, w)
\[ \text{\(\text{FOO}_x(y) := \text{Output \(x\) and halt.}\)} \]
\[ \text{\(\text{BAR}(M) := \text{Output "N(w) = Run FOO}_M\text{ outputting } M.\) } \]
\[ \text{\(\text{Run } M\text{ on }(M, w)"}\)} \]
\[ \text{\(\text{Q}(N, w) := \text{Run BAR}(N)\text{ outputting } S.\)} \]
\[ \text{\(\text{Run } T\text{ on }(S, w)\)} \]
\[ \text{\(\text{R}(w) := \text{Run FOO}_Q\text{ outputting } Q.\)} \]
\[ \text{\(\text{Run BAR}(Q)\text{ outputting } S.\)} \]
\[ \text{\(\text{Run } T\text{ on }(S, x)\)} \]

Claim: \(S\) is a description of \(R\) itself!
\[ \text{\(S(w) = \text{Run FOO}_Q\text{ outputting } Q.\)} \]
\[ \text{\(\text{Run BAR}(Q)\text{ outputting } S.\)} \]
\[ \text{\(\text{Run } T\text{ on }(S, w)\)} \]

Therefore \(R(w) = T(R, w)\)
For every computable $t$, there is a computable $r$ such that $r(w) = t(R,w)$ where

*R is a description of a TM computing r*

Moral: Suppose we can design a TM $T$ of the form

“On input $(x,w)$, do bla bla with $x$,
   do bla bla bla bla with $w$, etc. etc.”

We can always find a TM $R$ with the *behavior*:

“On input $w$, do bla bla with code of $R$,
   do bla bla bla bla with $w$, etc. etc.”

We can use the operation:

“*Obtain your own description*”

in Turing machine pseudocode!