Oracles

What if one could solve the halting problem?

New kind of Turing machine

How do I ask a question? Turing machine writes question on second tape and oracle reads question and gives yes or no answer.

The halting oracle answers the question "Does $M_i$ on $x_j$ halt?"

Let $S = \{(M, x) | M \text{ halts on } x\}$

Just as we defined recursive and r.e. we can define oracle recursive and oracle r.e. with respect to the oracle $S$.

1. (a) r.e. sets are oracle recursive  The oracle can tell if a Tm halts.
   (b) $L_D = \{x_i | x_i \notin L(M_i)\}$  oracle recursive since the oracle can solve the halting problem. $L_D$ is oracle recursive hard since $\{(M, x) | x \in L(M)\}$ is polynomial time reducible to $L_D$. Create $M_x$ that accepts input $M_x$ if $x \in L(M)$. Use $L_D$ to determine if $M_x \in L(M)$. 


2. $\exists$ oracle r.e. sets not oracle recursive. The sets of oracle Tm’s can be listed and we can by diagonalization define $L^\text{oracle}_D$.

There is a hierarchy of undecidable problems. Let $M$ be a regular Turing machine.

1. Is $x \in L(M)$? is r.e.
2. Does $L(M) = \emptyset$? equivalent to does there exists a valid computation.
3. Is $L(M)$ infinite?
4. Is $L(M)$ a regular set?
5. Is $L(M)$ a specific infinite set? $\forall x$ in set $\exists$ valid comp
6. Is $L(M)$ a regular set? $\exists$ set $\forall x$ in set $\exists$ valid comp.

**Oracles**

Attach an oracle for the halting problem to a Turing machine.

Define $M_S$ to be a Turing machine with an oracle for the set $S$.
Define $S_1 = \{M | L(M) = \emptyset\}$.

The set $S_1$ is not r.e. The complement set $\{M | L(M) \neq \emptyset\}$ is r.e.
Define $S_i = \{M^{S_{i-1}} | L(M^{S_{i-1}}) = \emptyset\}$.

The set $\{(M, x) | M \text{ halts when started on } x\}$ is equivalent to the complement of $S_1$. Equivalent means same complexity: the two sets are reducible to each other by an algorithm that halts on all inputs.

**Reductions**

$\{(M, x) | M \text{ halts on } x\}$ reduction to $\{M | L(M) \neq \emptyset\}$

To determine if $M$ halts on $x$ create $M_x$ that on every input simulates $M$ on $x$ and if $M$ halts then $M_x$ accepts its input. Thus

$$L(M_x) = \begin{cases} \Sigma^* & M \text{ halts on } x \\ \Phi & \text{otherwise} \end{cases}$$
\{M|L(M) \neq \Phi\} \text{ reduction to } \{(M,x)|M \text{ halts on } x\}

To determine if \(L(M) \neq \Phi\) reduction to "Mx halts on x" create \(M_x\) that on input \(x\) starts simulating \(M\) on longer and longer inputs for more and more steps looking for a string \(x\) that \(M\) accepts. Use the \((i,j)\) technique where one simulates the \(i^{th}\) Turing machine for \(j\) steps. If \(M\) accepts some string then \(M_x\) halts and accepts \(x\). Otherwise \(M_x\) runs forever.

The set \(\{M|L(M) = \Sigma^*\}\) is harder than membership. It is equivalent to \(S_2 = \{M^{S_1}|L(M^{S_1}) = \Phi\}\).

Reduction of \(\{M|L(M) = \Sigma^*\}\) to \(S_2 = \{M^{S_1}|L(M^{S_1}) = \Phi\}\).

Design \(M^{S_1}\) so that

\[
L(M^{S_1}) = \begin{cases} \Phi & \text{if } L(M) = \Sigma^* \\ \Sigma^* & \text{otherwise} \end{cases}
\]

If we could determine if \(L(M^{S_1}) = \Phi\), then we could determine if \(L(M) = \Sigma^*\). \(M^{S_1}\) on every input looks for \(x\) not in \(L(M)\). If \(M^{S_1}\) finds an \(x\) not in \(L(M)\), then \(M^{S_1}\) accepts its input. To find \(x\) not in \(L(M)\) requires the halting problem.

Reduction of \(S_2 = \{M^{S_1}|L(M^{S_1}) = \Phi\}\) to \(\{M|L(M) = \Sigma^*\}\). To determine if \(L(M^{S_1}) = \Phi\) create \(M\) that accepts all invalid computations of \(M^{S_1}\). If the set of invalid computations is \(\Sigma^*\) then \(L(M^{S_1}) = \Phi\).

A valid computation of \(M^{S_1}\) has \(q?\) inserted when \(M^{S_1}\) asks a question followed by a valid computation of an ordinary Turing machine followed by \(q_y\) if the answer is yes and just the symbol \(q_n\) if the answer is no since one needs an oracle when the answer is no. The only complicated issue in checking that the string is an invalid computation is when the oracle returns a no. In this case the oracle has said that \(L(M) = \Phi\) when actually accepting some string. One can determine this by the \((i,j)\) technique of simulating \(M\) on \(x_i\) for \(j\) steps. Use oracle to determine if process halts.

The set \(\{M|L(M) \text{ is a regular set}\}\) is equivalent to \(S_3\).

A hierarchy

\[
L(M) \neq \Phi \quad \exists x \ x \in L(M) \quad \exists \text{ valid comp} \\
L(M) = \Phi \quad \forall x \ x \notin L(M) \quad \forall \text{ strings not valid comps} \\
L(M) \neq \Sigma^* \quad \exists x \ \forall \text{ comps comp not valid} \\
L(M) = \Sigma^* \quad \forall x \ \exists \text{ valid comp} \\
L(M) \text{ regular set} \quad \exists \text{ regular set } \forall x \ x \text{ in } R \text{ iff } x \text{ in } L(M)
\]
1. (a) \(\{M | L(M) \neq \Phi\}\) is r.e. Run \(M\) on longer and longer inputs for more and more steps looking for \(x\) that is accepted.

(b) \(\{M | L(M) \neq \Phi\}\) is complete for r.e. One can answer does \(M\) accept \(x\) for any Turing machine \(M\). Create \(M_x\) where \(L(M_x) = \begin{cases} \Sigma^* & \text{if } M \text{ accepts } x \\ 0 & \text{otherwise} \end{cases}\)

2. (a) \(\{M | L(M) = \Phi\}\) is \(S_1\) recursive Use oracle to determine whether or not \(M\) in \(S_1\).

(b) \(\{M | L(M) = \Phi\}\) is \(S_1\) complete

3. (a) \(\{M^{S_1} | L(M^{S_1}) \neq \Phi\}\) is \(S_1\) r.e. Run \(M^{S_1}\) on longer and longer inputs for more and more steps looking for \(x\) that is accepted.

(b) \(\{M | L(M) \neq \Sigma^*\}\) is \(S_1\) r.e. Use oracle for \(\{[M, x] | x \in L(M)\}\) which is equivalent to \(S_1\) to search for \(x \neq L(M)\). If it finds \(x \notin L(M)\) it halts. Otherwise it runs forever.
4. (a) $\{M^{S_1}|L(M^{S_1}) = \Phi\}$ is $S_2$ recursive  Use oracle for $S_1$ to determine whether or not $M^{S_1}$ in $S_1$.

(b) $\{M|L(M) = \Sigma^*\}$ is $S_2$ recursive  Create $M^{S_1}$ that searches for $x \notin L(M)$. The $S_1$ oracle answers if a given $x$ is or is not in $L(M)$. If $M^{S_1}$ finds an $x$ not in $L(M)$ it stops, otherwise it runs forever. Use $S_2$ to determine if $M^{S_1}$ stops.

5. (a) $\{M^{S_2}|L(M^{S_2}) \neq \Phi\}$ is $S_2$ r.e. Run $M^{S_2}$ on longer and longer inputs for more and more steps looking for $x$ that is accepted.

(b) $\{M^{S_1}|L(M^{S_1}) \neq \Sigma^*\}$ is $S_2$ r.e. Look at longer and longer $x$ using $S_2$ to determine if $x \in L(M^{S_1})$. If we find string halt and say yes. Otherwise run forever.

(c) $\{M|L(M) is a regular set\}$ is $S_2$ r.e. Cycle through all regular sets.
   For each regular set $R$ ask if $L(M) = R$.
   To do this cycle through all $x$
      ask if $x$ is $L(M)$ using $S_1$.
      ask if $x$ is found proving $L(M) \neq R$ by $S_2$. If yes move on to next $R$ otherwise return yes $L(M)$ is regular.
   If regular set found we return yes. Else we run forever. Thus $S_2$ r.e.