A reduction from set $A$ to set $B$ is a computable function $\sigma : A \rightarrow B$ such that

$$x \in A \iff \sigma(x) \in B.$$ 

If $A$ reduces to $B$ via $\sigma$ we write $A \leq_{\sigma} B$.

We use reductions to prove certain sets are not recursive or not r.e. In class we showed:

1. If $A \leq B$ and $A$ is not recursive, then $B$ is not recursive.
2. If $A \leq B$ and $A$ is not r.e., then $B$ is not r.e.

So if we have a set $A$ that we know to be not recursive (e.g., $\overline{HP} = \{M\#x \mid M \text{ loops on } x\}$) and we can reduce that set $A$ to another set $B$, then we have shown $B$ is not recursive. Similarly a reduction from a non-r.e. set $A$ (e.g., $\overline{HP} = \{M\#x \mid M \text{ loops on } x\}$) to another set $B$ would show that $B$ is not r.e.

**Reduction Examples:**

1. Let $ALL = \{M \mid L(M) \text{ accepts all strings}\}$. Show $ALL$ is not recursive.

   **Solution:** We know the set $HP = \{M\#x \mid M \text{ halts on } x\}$ is r.e. but not recursive. We show $HP \leq ALL$, and therefore that $ALL$ is not recursive:

   Given TM $M$ and input string $x$, we describe TM $N = \sigma(M\#x)$ such that

   $$M\#x \in HP \iff N \in ALL.$$ 

   That is, given TM $M$ and input string $x$, we describe TM $N = \sigma(M\#x)$ such that

   $$M \text{ halts on } x \iff N \text{ accepts all strings}.$$ 

   TM $N$ will have $M$ and $x$ hardcoded into its design. TM $N$ on its input $y$ does the following:

   (a) Simulates $M$ on $x$.
   
   (b) If $M$ halts on $x$ then $N$ accepts $y$.

   Our description of $N$ ensures that if $M$ halts on $x$, then $N$ accepts all strings; if $M$ loops on $x$, then $N$ accepts $\emptyset \neq \Sigma^*$. This description of $N$ is exactly the behavior we needed to achieve a reduction $HP \leq ALL$. Thus we have shown $ALL$ is not recursive.
2. Let \( \text{REC} = \{ M \mid L(M) \text{ is recursive} \} \). Show \( \text{REC} \) is not r.e.

**Solution:** We know the set \( \overline{\text{HP}} = \{ M\#x \mid M \text{ loops on } x \} \) is not r.e.

We show \( \overline{\text{HP}} \preceq \text{REC} \), and therefore that \( \text{REC} \) is not r.e.: Given TM \( M \) and input string \( x \), we describe TM \( N = \sigma(M\#x) \) such that

\[
M\#x \in \overline{\text{HP}} \iff N \in \text{REC}.
\]

That is, given TM \( M \) and input string \( x \), we describe TM \( N = \sigma(M\#x) \) such that

\[
M \text{ loops on } x \iff L(N) \text{ is recursive}.
\]

TM \( N \) will have \( M \) and \( x \) hardcoded into its design. TM \( N \) on its input \( y \) does the following:

(a) Simulates \( M \) on \( x \).

(b) If \( M \) halts on \( x \) then \( N \) runs machine \( K \) on \( y \), where \( K \) is a TM that accepts the non-recursive set \( \text{HP} \). \( N \) accepts \( y \) iff \( K \) accepts \( y \).

Our description of \( N \) ensures that if \( M \) loops on \( x \), then \( N \) accepts \( \{ \} \), a recursive set; if \( M \) halts on \( x \), then \( N \) accepts \( \text{HP} \), a non-recursive set. This description of \( N \) is exactly the behavior we needed to achieve a reduction \( \overline{\text{HP}} \preceq \text{REC} \). Thus we have shown \( \text{REC} \) is not r.e.

3. Let \( \text{INF} = \{ M \mid L(M) \text{ is infinite} \} \). Show \( \text{INF} \) is not r.e.

**Solution:** We know the set \( \overline{\text{HP}} = \{ M\#x \mid M \text{ loops on } x \} \) is not r.e.

We show \( \overline{\text{HP}} \preceq \text{INF} \) and therefore that \( \text{INF} \) is not r.e.: Given TM \( M \) and input string \( x \), we describe TM \( N = \sigma(M\#x) \) such that

\[
M\#x \in \overline{\text{HP}} \iff N \in \text{INF}.
\]

That is, given TM \( M \) and input string \( x \), we describe TM \( N = \sigma(M\#x) \) such that

\[
M \text{ loops on } x \iff N \text{ accepts an infinite set}.
\]

TM \( N \) will have \( M \) and \( x \) hardcoded into its design. TM \( N \) on its input \( y \) does the following:

(a) Simulates \( M \) on \( x \) for \( |y| \) steps.

(b) If \( M \) halts on \( x \) in \( |y| \) steps, then \( N \) rejects \( y \). Otherwise if \( M \) has not yet halted on \( x \), then \( N \) accepts \( y \).

Our description of \( N \) ensures that if \( M \) loops on \( x \), \( N \) accepts \( \Sigma^* \), an infinite set; if \( M \) halts on \( x \), then \( N \) accepts \( \{ y \mid |y| < \text{number of steps that } M \text{ runs on } x \} \), a finite set. This description of \( N \) is exactly the behavior we needed to achieve a reduction \( \overline{\text{HP}} \preceq \text{INF} \). Thus we have shown \( \text{INF} \) is not r.e.
4. Let $REC = \{M \mid L(M) \text{ is recursive}\}$. Show $REC$ is not recursive.

**Solution:** We know the set $HP = \{M\#x \mid M \text{ halts on } x\}$ is r.e. but not recursive. We show $HP \leq REC$, and therefore that $REC$ is not recursive:

Given TM $M$ and input string $x$, we describe TM $N = \sigma(M\#x)$ such that

$$M\#x \in HP \iff N \in REC.$$  

That is, given TM $M$ and input string $x$, we describe TM $N = \sigma(M\#x)$ such that

$$M \text{ halts on } x \iff N \text{ accepts a recursive set.}$$

TM $N$ will have $M$ and $x$ hardcoded into its design. TM $N$ on its input $y$ does the following:

(a) Simulates $M$ on $x$ for $|y|$ steps.

(b) If $M$ halts on $x$ in $|y|$ steps, then $N$ rejects $y$. Otherwise if $M$ has not yet halted on $x$, then $N$ runs machine $K$ on $y$, where $K$ is a TM that accepts the non-recursive set $HP$. $N$ accepts $y$ iff $K$ accepts $y$.

Our description of $N$ ensures that if $M$ halts on $x$, then $N$ accepts the set of strings

$$\{y \mid |y| < \text{number of steps that } M \text{ runs on } x\},$$

a finite and therefore recursive set; if $M$ loops on $x$, then $N$ accepts $HP$, a non-recursive set. This description of $N$ is exactly the behavior we needed to achieve a reduction $HP \leq REC$. Thus we have shown $REC$ is not recursive.

5. Let $A$ and $B$ be sets of Turing machines such that

$$A = \{M \mid L(M) = \Sigma^*\} \text{ and } B = \{N \mid L(N) = \{0^n1^n \mid n \geq 1\}\}.$$  

Show that $A \leq B$.

Given TM $M$ we describe TM $N = \sigma(M)$ such that

$$M \in A \iff N \in B.$$  

That is, given TM $M$ we describe TM $N = \sigma(M)$ such that

$$M \text{ accepts } \Sigma^* \iff N \text{ accepts } \{0^n1^n \mid n \geq 1\}.$$  

TM $N$ will have $M$ hardcoded into its design. TM $N$ on its input $y$ does the following:

(a) Checks whether $y$ is of the form $0^n1^n$; if not, reject.

(b) Simulates $M$ on all strings $x$ where $|x| \leq |y|$ and accepts iff $M$ accepts all of them.

Our description of $N$ ensures that if $M$ accepts $\Sigma^*$ then $N$ accepts all string of the form $0^n1^n$; if $M$ does not accept $\Sigma^*$ then $N$ will not accept all strings of the form $0^n1^n$. This description of $N$ is exactly the behavior we needed to achieve a reduction $A \leq B$. 